GH. DODESCU D. IONESCU
LIVIA NISIPEANU F. PILAT

LIMBAJUL BASIC ȘI APLICAȚII

EDITURA
DIDACTICĂ
ȘI
PEDAGOGICĂ
BUCUREȘTI - 1978

MINISTERUL EDUCAŢIEI ŞI ÎNVĂŢĂMÎNTULUI

CONF. DR. ING. GH. DODESCU
ASIST. ING. LIVIA NISIPEANU
LECT. ING. D. IONESCU
LECT. DR. ING. F. PILAT

LIMBAJUL BASIC ŞI APLICAŢII

EDITURA DIDACTICĂ ŞI PEDAGOGICĂ
BUCUREŞTI — 1978

Referent ştiinţific: conf. dr. **Călin Ignat**

Contribuţia autorilor:

conf. dr. ing. **G. Dodescu** cap. X, XIV, XV, XVI, XVII
lect. ing. **D. Ionescu** cap. I, II, III
asist. ing. **L. Nisipeanu** cap. IV, V, VI, VII, VIII, IX
lect. dr. ing. **F. Pilat** cap. XI, XII, XIII

Redactor: *Gabriela Iliescu*
Tehnoredactor: *Achille Daniel*
Grafician: *Nicolae Sîrbu*

PREFAȚĂ

Ampla dezvoltare pe care o cunoaște dotarea cu tehnică de calcul modernă a economiei naționale precum și utilizarea acesteia în activitatea de învățămînt, cercetare și proiectare, reclamă o aprofundare rapidă a cunoștințelor de programare. În acest context se înscrie lucrarea de față. care prezintă pentru prima dată la noi în țară un limbaj conversațional.

În cele 17 capitole sînt prezentate elementele și structura limbajului, modul de lucru cu fișierele, modalitățile de utilizare ale consolei precum și un mare număr de aplicații.

Caracterul aplicativ al lucrării este evidențiat atît prin exemplele și exercițiile aferente fiecărui capitol cît și prin cele 33 de programe rulate pentru rezolvarea unor probleme din domeniul matematicii, economiei etc., prezentate în capitolele XIV—XVII.

Lucrarea se adresează studenților de la facultățile care au în planul de învățămînt disciplina de programare, cadrelor didactice, economiștilor, inginerilor precum și tuturor celor care doresc să-și realizeze lucrările de cercetare și proiectare cu ajutorul tehnicii de calcul, făcînd uz de un limbaj simplu, elastic și ușor de asimilat.

În încheiere mulțumim conducerii catedrei de Cibernetică economică pentru climatul favorabil de cercetare creat și pentru tehnica de calcul pusă la dispoziție, precum și colegilor de la celelalte catedre pentru sprijinul acordat în formularea problemelor aplicative.

AUTORII

TABLA DE MATERIE

Capitolul I

INTRODUCERE

Limbajul BASIC a fost elaborat de către un colectiv aflat sub îndrumarea profesorului J. G. Kemeny, la colegiul Dartmouth (S.U.A.), în anul 1965, [40]. Prima versiune a fost implementată pe un calculator General Electric 225. Numele limbajului este un acronim care provine de la cuvintele din limba engleză „*B*eginner's *A*ll-Purpose *S*ymbolic *I*nstruction *C*odes", care în traducere înseamnă: cod simbolic de instrucţiuni, de scop general, pentru începători.

La scurt timp după apariţia sa limbajul a devenit foarte popular datorită unor calităţi reale ce l-au impus. Dintre acestea enumerăm:

a) Simplitatea limbajului. Limbajul nu impune atît de multe restricţii şi convenţii ca alte limbaje consacrate (FORTRAN, COBOL, ALGOL), în schimb păstrează posibilităţile de lucru ale acestora (utilizarea de masive de date, de fişiere, etc.), avînd în unele privinţe şi facilităţi suplimentare, cum ar fi lucrul cu matrice într-o formă agregată.

b) Este un limbaj de tip conversaţional care permite interacţiunea utilizatorului cu programul.

c) Oferă facilităţi în aplicarea tehnicilor avansate de prelucrare automată a datelor cum ar fi "time-sharing" (acces multiplu) şi "teleprocessing" (teleprelucrare).

d) Permite cu uşurinţă aplicarea sistemului de lucru în timp real, sistem ce aduce mari avantaje utilizatorilor.

Pentru a înţelege mai bine avantajele ce decurg din aceste atribuţii este necesară o descriere a lor mai detaliată.

La primele calculatoare electronice programatorul îşi introducea programul în cod maşină de la claviatura unei maşini de scris şi urmărea direct desfăşurarea lui intervenind pe loc în cazul apariţiei unor erori. S-a constatat însă că acest mod de lucru este neeconomic deoarece o mare parte din timp calculatorul nu era utilizat aşteptînd mesaje de la consolă. Exista o inadvertenţă flagrantă între viteza de calcul a maşinii (sute de mii de operaţii pe secundă) şi viteza cu care programatorul îi furniza datele de lucru.

Acest neajuns a fost însă remediat trecîndu-se la tehnici de exploatare avansate prin care s-a urmărit optimizarea utilizării resurselor echipamentului de calcul. Au apărut sistemele de operare şi s-a impus o anumită disciplină privind accesul programelor la calculator. Astfel beneficiarii îşi trec programele pe un suport tehnic (cartele sau bandă perforată), în regim „off line" şi le predau apoi la un serviciu de dispecerat care urmează să le lanseze în execuţie conform unei planificări prestabilite. Unul sau mai multe programe de utilizator constituie o lucrare (job) şi reprezintă o unitate de execuţie pentru care se alocă resursele fizice şi logice necesare. Un astfel de acces la calculator, realizat sub forma unui şir de lucrări poartă denumirea de exploatare secven-

ţială sau serială. În vorbirea curentă se utilizează foarte adesea termenul de limbă engleză „batch processing" sau prelucrare pe loturi.

Aplicarea sistemului de exploatare „batch processing" reprezintă o optimizare din punct de vedere al utilizării resurselor fizice şi logice ale sistemului de calcul dar foarte adesea aduce prejudicii beneficiarilor. Prin natura sa, acest sistem de lucru izolează complet utilizatorul de programele sale pe perioada executării lucrării. Pe de altă parte accesul prin planificare face ca timpul de răspuns* să fie nedefinit putînd varia de la cîteva ore pînă la cîteva zile.

Imposibilitatea asistării de către utilizator a lucrării pe timpul executării acesteia apare ca un inconvenient încă din faza de testare a programelor. Pentru a ilustra acest fapt este necesară o descriere sumară a fazelor prin care trece programul din momentul elaborării sale pînă la execuţie şi obţinerea rezultatelor. Să urmărim pentru aceasta figura 1.1.

Presupunem un program scris în limbajul FORTRAN şi prelucrat în modul secvenţial.

Utilizatorul îşi scrie instrucţiunile şi datele pe foi de programare, trece apoi la perforarea acestora pe cartele (în regim „off line") şi predă programul la dispecerat pentru execuţie. Urmează un timp de aşteptare impus de planificarea lucrărilor, după care programul este încărcat şi supus compilării pentru a fi transmis în limbaj maşină.

În urma compilării, în marea majoritate a cazurilor rezultă erori de sintaxă care sînt afişate la imprimantă şi lucrarea se opreşte în acest punct. Utilizatorul primeşte lista de erori, le analizează şi introduce corecţiile necesare în pachetul de cartele, apelînd prin planificare la un nou acces la calculator. Dacă unele erori persistă şi la a doua rulare este necesară repetarea procesului, rezultînd în final un timp destul de mare cheltuit de către utilizator pînă la corectarea tuturor erorilor de sintaxă.

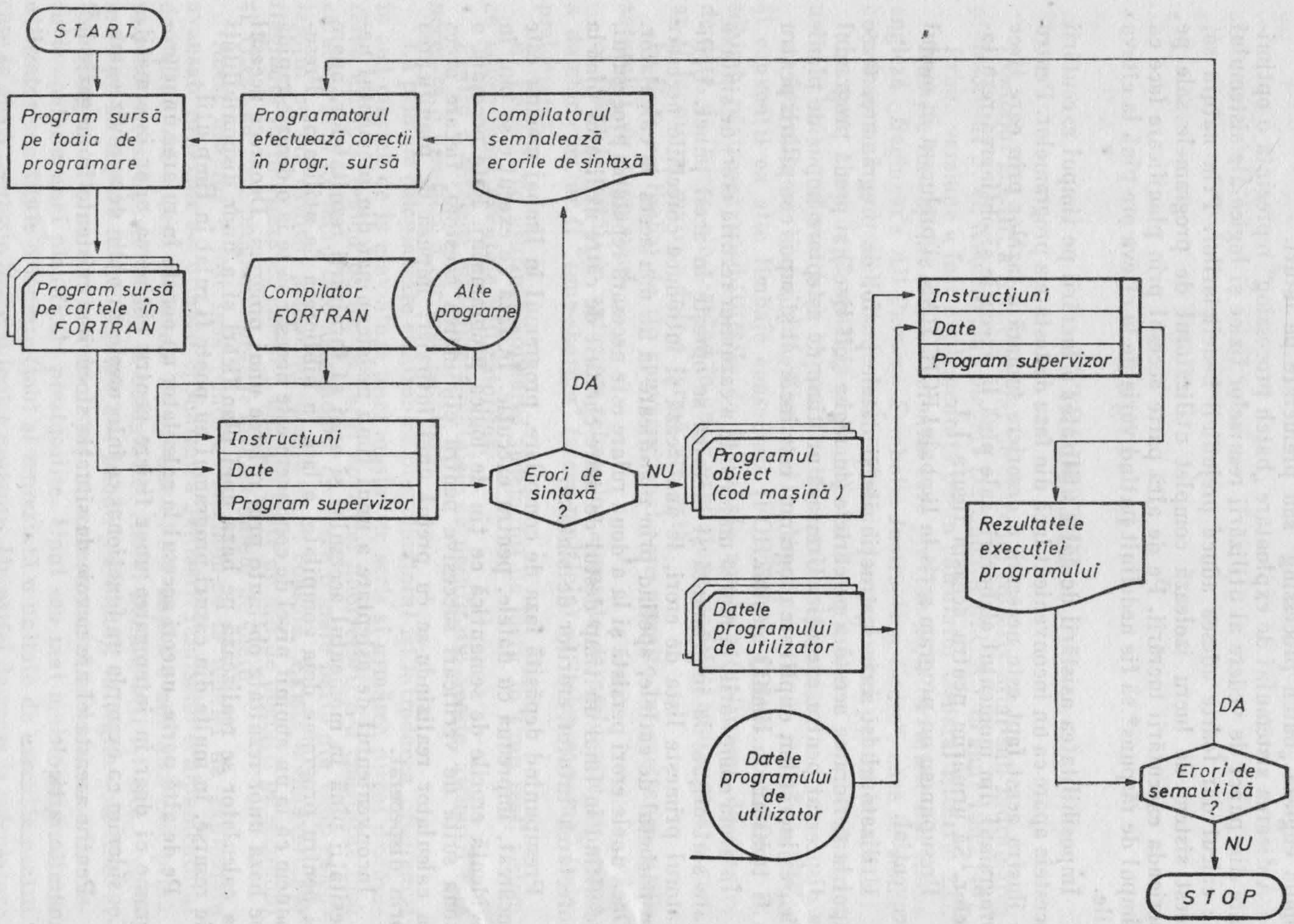
Presupunînd depăşită faza de compilare, programul în limbaj maşină este încărcat, împreună cu datele, pentru execuţie. În faza de execuţie se pun în evidenţă erorile de semantică ce ţin de logica programului. Este necesară o nouă suită de verificări succesive pentru eliminarea acestora, fiecare acces la calculator realizîndu-se cu preţul unei întîrzieri impusă de planificarea prin dispecerat.

Inconvenientul de aşteptare a unui timp nedeterminat din momentul necesităţii pînă în momentul accesului şi apoi al furnizării rezultatelor, apare şi pentru programe deja compilate, aflate în biblioteca de utilizator. Presupunem că la un anumit nivel de conducere este necesar să se ia o decizie rapidă pe baza unor rezultate obţinute prin rularea unui program. Deoarece accesul la calculator se realizează pe baza unei planificări şi a unor disponibilităţi de resurse, în multe din cazuri programul nu poate fi rulat în timp util.

Pe de altă parte, uneori accesul la calculator nu constă în rularea unor programe ci doar în interogarea unor fişiere pentru obţinerea unor informaţii. Considerăm ca exemplu un funcţionar ce întocmeşte un ordin de aprovizionare.

Pentru aceasta el are nevoie de situaţia stocurilor existente în magazie la anumite articole.

*** Timpul de răspuns este intervalul de timp între momentul apariţiei unui mesaj de prelucrare şi momentul răspunsului sistemului la acest mesaj.**



START
Program sursă pe foaia de programare
Programatorul efectuează corecții în progr. sursă
Compilatorul semnalează erorile de sintaxă
Program sursă pe cartele în FORTRAN
Compilator FORTRAN
Alte programe
DA
Instrucțiuni
Date
Program supervizor
Erori de sintaxă ?
NU
Programul obiect (cod mașină)
Instrucțiuni
Date
Program supervizor
Rezultatele execuției programului
Datele programului de utilizator
Datele programului de utilizator
DA
Erori de semautică ?
NU
STOP

Fig. 1.1.

Interogarea fişierului de inventar, aflat la centrul de calcul, nu se poate face decît în baza unei planificări, lucru ce îl ţine pe loc în întocmirea ordinului respectiv.

Se pot găsi multe astfel de exemple pentru a ilustra urmările unui acces greoi la calculator, rezultat din sistemul de prelucrare secvenţială (batch processing). E suficient să amintim probleme ca: rezervarea de locuri în transportul aerian şi feroviar, rezervarea de locuri la hoteluri, interogarea unor fişiere pentru luarea unor decizii rapide în conducerea întreprinderilor sau a ramurilor economice, dispecerat energetic, prelucrarea observaţiilor meteorologice, gestiunea automată a bibliotecilor etc.

Dacă unele dintre aplicaţii suferă serios de pe urma prelucrării secvenţiale, altele nici nu pot fi măcar implementate într-un astfel de sistem din cauza unui timp de răspuns critic ce le caracterizează. În această categorie intră toate aplicaţiile de conducere cibernetică a unor procese tehnologice, activităţi economice, activităţi militare, etc. Pentru realizarea conducerii şi controlului în sens cibernetic a unor procese — potrivit unui program prescris sau unor valori de optim ale parametrilor specifici procesului — este necesar să se măsoare valorile instantanee ale parametrilor procesului şi să fie prelucrate în timp util iar rezultatele să fie redate procesului în vederea reglării acestuia sau în vederea luării unor decizii operative, [11], [22].

Realizarea unui acces permanent la calculator, primirea rezultatelor de la acesta într-un interval de timp prestabilit şi posibilitatea de a interveni în program pe timpul executării acestuia, sînt facilităţi de mare importanţă pentru utilizator şi definesc împreună un sistem de lucru conversaţional în timp real.

Pentru implementarea unui astfel de sistem sînt necesare resurse „hardware“ şi „software“ corespunzătoare. Din punct de vedere software este necesar un sistem de operare în timp real şi un limbaj de programare de tip conversaţional (limbajul BASIC este un astfel de limbaj).

Din punct de vedere „hardware“ se impune în primul rînd existenţa unor periferice de tip interactiv şi apoi a unor resurse fizice suficiente pentru realizarea unui timp de răspuns destul de mic impus de utilizator prin noţiunea de „timp real“. Ca periferice interactive, cele mai utilizate în prezent sînt maşina de scris (typewriter) şi dispozitivul de afişaj cu tub catodic (display) înzestrat cu claviatură.

În privinţa volumului de resurse impus de timpul de răspuns, ar fi ideal ca fiecare utilizator să aibă calculatorul său propriu. Acest lucru nu este însă posibil deoarece este prea costisitor, motiv pentru care s-a imaginat o nouă tehnică de exploatare a resurselor denumită „time-sharing“ [57]. Prin time-sharing (în traducere directă „divizarea timpului“) se înţelege împărţirea timpului de ocupare a unor resurse fizice şi logice (inclusiv timpul unităţii centrale de prelucrare) între mai mulţi utilizatori. Pentru a putea lucra în timp real sistemul trebuie însă astfel proiectat încît împărţirea resurselor să fie transparentă pentru utilizatori, fiecare avînd impresia că este singur în raport cu calculatorul. Acest lucru este posibil în baza faptului că viteza de lucru a sistemului este mult mai mare decît viteza de lucru a utilizatorului la periferice, ceea ce face posibil ca în timp ce unii utilizatori îşi introduc lucrările, sistemul să prelucreze şi să afişeze răspunsurile lucrărilor altor utilizatori.

Tehnica de exploatare în „time-sharing“ s-a dezvoltat în paralel cu alte facilităţi oferite beneficiarilor, facilităţi înglobate în conceptul de „telepre-

lucrare" (teleprocessing) [30]. În teleprelucrare utilizatorul are acces la calculator de la un periferic ce nu se mai află în sala calculatorului ci într-o altă sală, altă clădire sau altă localitate. Legătura la calculator se realizează prin linii de telecomunicații specifice telefoniei urbane sau interurbane. Prin aceasta utilizatorul poartă dialog cu sistemul de calcul direct de la locul său de muncă, nefiind obligat să se deplaseze în sala calculatorului. Echipamentul aflat în dotarea utilizatorului poartă denumirea de terminal și se compune în general dintr-un periferic interactiv și unul sau mai multe din următoarele periferice: cititor/perforator de bandă, cititor/perforator de cartele, imprimantă precum și echipament de adaptare la linie denumit modem (modulator-demodulator). Dacă terminalul este axat pe urmărirea unui proces tehnologic în componența sa vor intra obligatoriu convertoare de tip analogic -digital și digital-analogic.

Teleprelucrarea este legată în special de modalitățile de lucru în timp real și „time-sharing" dar nu este exclusă posibilitatea implementării ei într-un sistem de lucru secvențial, cînd poartă denumirea de „remote batch processing" sau „remote job entry", care în traducere înseamnă „prelucrarea pe loturi de lucrări (secvențial), de la distanță".

În acest caz nu mai este absolut necesară prezența perifericului interactiv la terminal, fiind suficiente cititoare/perforatoare de bandă, cititoare/perforatoare de cartele (și) sau imprimante.

Modemul realizează o transformare a semnalelor electrice din forma în care sînt emise de terminal (impulsuri rectangulare) la forma în care sînt acceptate de linie (semnale analogice) și invers. El reprezintă deci un echipament de adaptare a terminalului la linie. Pentru distanțe scurte acest echipament de adaptare poate lipsi.

Noțiunea de distanță scurtă se precizează de la caz la caz în funcție de echipamentul terminal utilizat (mai ales în funcție de viteza de transmisie în linie a acestuia) și de calitatea canalului de telecomunicații la care e legat terminalul.

În cadrul aplicațiilor de teleprelucrare, o componentă importantă o constituie legătura de telecomunicație necesară între terminal și stația centrală de calcul. Din punct de vedere al modului în care se stabilesc, aceste legături sînt de două tipuri: permanente și comutate, [66].

Legăturile permanente se stabilesc pe linii proprii sau închiriate din rețeaua telefonică și se caracterizează prin faptul că legătura se realizează pe aceeași linie fizică ori de cîte ori utilizatorul de la terminal face apel la serviciile calculatorului.

Legăturile comutate se stabilesc prin rețeaua comutată, deci la fiecare utilizare a calculatorului legătura se realizează pe alte circuite fizice, în funcție de trunchiurile găsite libere în centrala telefonică la momentul respectiv.

Pentru a putea lucra cu calculatorul, utilizatorul de la terminal va trebui să stabilească mai întîi legătura fizică și abia apoi legătura logică.

Același lucru e valabil și atunci cînd calculatorul dorește să stabilească o legătură cu un terminal.

Legătura fizică se realizează prin formarea unui număr la un disc sau la o tastatură telefonică, în urma căreia are loc o convorbire telefonică ce confirmă stabilirea legăturii. Apoi se comută pe transmitere de date și terminalul intră într-o conexiune logică cu calculatorul. Timpul cît terminalul este legat logic

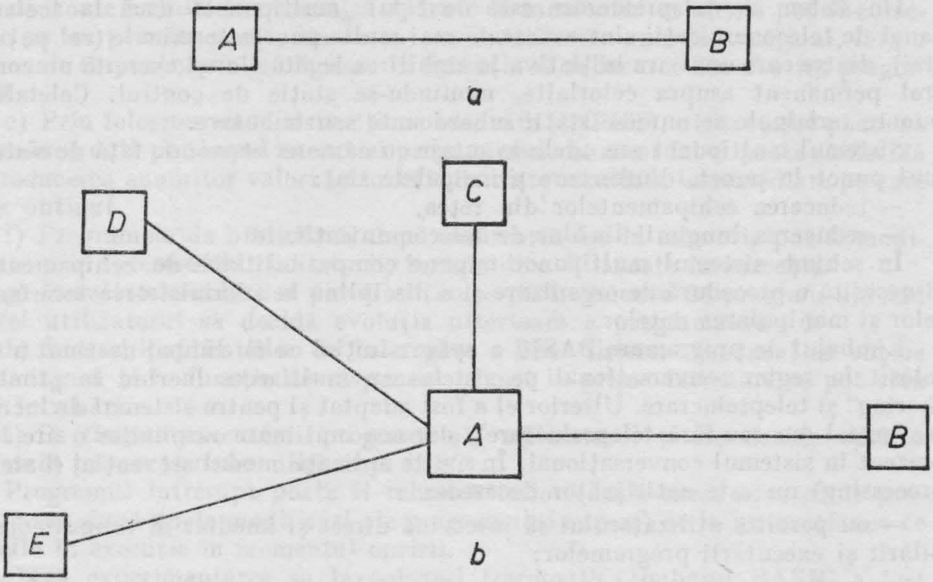


Fig. 1.2.

la calculatorul central, timp în care utilizatorul transmite programe, date şi mesaje şi primeşte rezultatele prelucrărilor, se numeşte sesiune de terminal.

În unele cazuri există şi posibilităţi de apel automat, cînd se realizează simularea numărului prin program, sau de răspuns automat, cînd nu mai are loc o convorbire telefonică ci se trece automat pe transmiterea de date, iar chemătorul primeşte confirmarea stabilirii legăturii sub forma unui ton.

Este evident că legăturile permanente sînt mai sigure decît cele comutate dar în acelaşi timp sînt şi mai scumpe deoarece linia stă la dispoziţia utilizatorului, 24 de ore din 24, chiar dacă acesta nu o foloseşte integral. În cazul legăturilor comutate utilizatorul plăteşte pe bază de contor, deci numai pe timpul cît lucrează efectiv.

O altă clasificare importantă în cazul sistemelor de teleprelucrare este aceea făcută în funcţie de numărul punctelor terminale între care se face transmisiunea de date prin acelaşi canal de telecomunicaţii. Din acest punct de vedere există sisteme de teleprelucrare de tipul punct la punct (fig. 1.2) şi sisteme de tipul multipunct (fig. 1.3), [32].

Un sistem de teleprelucrare este de tipul „punct la punct" dacă la canalul de telecomunicaţii sînt conectate numai două puncte terminale, la ambele terminaţii ale canalului existînd cîte un echipament de transmisie.

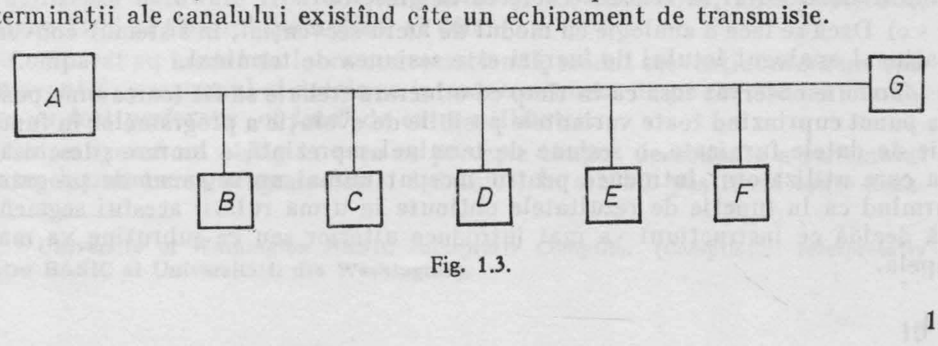


Fig. 1.3.

Un sistem de teleprelucrare este de tipul „multipunct" dacă la acelaşi canal de telecomunicaţii sînt conectate mai multe puncte terminale (cel puţin trei), dintre care unul are iniţiativa în stabilirea legăturilor şi exercită un control permanent asupra celorlalte, numindu-se staţie de control. Celelalte puncte terminale se numesc staţii subordonate sau tributare.

Sistemul multipunct are unele avantaje cu caracter economic faţă de sistemul punct la punct, dintre care principalele sînt:

— reducerea echipamentelor din reţea,

— reducerea lungimii liniilor de telecomunicaţii.

În schimb sistemul multipunct impune compatibilitatea de echipamente şi necesită o procedură de organizare şi o disciplină în administrarea terminalelor şi manipularea datelor.

Limbajul de programare BASIC a apărut iniţial ca un limbaj destinat a fi folosit în regim conversaţional pe sisteme cu multiacces lucrînd în „time-sharing" şi teleprelucrare. Ulterior el a fost adaptat şi pentru sistemul de lucru secvenţial (cu sau fără teleprelucrare), dar cea mai mare răspîndire o are în prezent în sistemul conversaţional. În multe aplicaţii modul secvenţial (batch processing) nu este satisfăcător deoarece:

— nu permite utilizatorului să intervină direct şi imediat în timpul compilării şi executării programelor;

— conducerea operativă a unor procese şi luarea deciziilor impun timpi de răspuns foarte mici şi deci utilizarea calculatorului în timp real;

— în aplicaţii referitoare la exploatarea băncilor de date sau a unor fişiere operative, utilizatorul trebuie să poată consulta pe loc baza de date şi utiliza imediat biblioteca de programe.

Limbajul BASIC a fost conceput special pentru a putea fi folosit în regim conversaţional şi în aplicaţii de timp real. Pentru utilizarea sa este necesar însă un software corespunzător care trebuie să cuprindă în primul rînd compilatoare şi interpretatoare conversaţionale şi incrementale, care permit o acţiune directă între utilizator şi calculator la nivelul limbajului simbolic, atît pentru compilare cît şi pentru execuţie.

Iată cîteva din principalele aspecte ale interacţiunii utilizator-program pe perioada introducerii, compilării şi execuţiei acestuia din urmă:

a) Pe timpul introducerii instrucţiunilor şi datelor din claviatură, utilizatorul poate să şteargă sau să modifice caractere sau instrucţiuni întregi.

b) După introducerea unei instrucţiuni din claviatură se acţionează o tastă (RETURN) care determină activarea instrucţiunii respective în memoria calculatorului central. Are loc compilarea imediată a instrucţiunii şi în cazul existenţei unor erori de sintaxă se afişează deîndată mesajul de eroare, permiţînd utilizatorului să realizeze corectarea greşelilor.

c) Dacă se face o analogie cu modul de lucru secvenţial, în sistemul conversaţional analogul lotului de lucrări este sesiunea de terminal.

Trebuie observat însă că în timp ce o lucrare trebuie să fie foarte bine pusă la punct cuprinzînd toate variantele posibile de evoluţie a programelor în funcţie de datele furnizate, o sesiune de terminal reprezintă o lucrare „deschisă" la care utilizatorul introduce pentru început numai un segment de program urmînd ca în funcţie de rezultatele obţinute în urma rulării acestui segment să decidă ce instrucţiuni va mai introduce ulterior sau ce subrutine va mai apela.

d) Toate erorile de semantică apărute la momentul execuţiei pot fi corectate pe loc prin înserarea, anularea sau modificarea unor instrucţiuni, fără a fi nevoie ca pentru o nouă lansare în execuţie să se încarce din nou tot programul.

e) Prin folosirea unor instrucţiuni adecvate (INPUT), utilizatorul poate să introducă date pe timpul execuţiei programului. În acest fel el poate să decidă introducerea anumitor valori în funcţie de nişte rezultate intermediare pe care le-a obţinut.

f) Programele de bibliotecă, după încărcarea lor în memorie pot fi modificate prin intermediul claviaturii, înainte de a fi lansate în execuţie.

g) Rezultatele rulărilor pot fi obţinute pe fragmente de program ajutînd astfel utilizatorul să decidă evoluţia ulterioară a programului.

h) În cazul aplicaţiilor de interogare a unor fişiere, răspunsul se obţine instantaneu şi în funcţie de răspunsul primit la prima întrebare se pot crea noi întrebări.

i) Pe timpul executării programului, utilizatorul poate decide oprirea acestuia prin acţionarea unei taste (ESC).

Programul întrerupt poate fi reluat ulterior (după executarea altui program sau după unele modificări ale programului curent) de la instrucţiunea ce se afla în execuţie în momentul opririi.

După experimentarea sa la colegiul Darmouth, limbajul BASIC a fost comercializat de firma General Electric care şi-a axat producţia de calculatoare pe sisteme cu acces multiplu. Ulterior şi alte firme au construit compilatoare de BASIC, făcînd limbajul accesibil. Dintre acestea menţionăm: Data General, Scientific Data Systems, Borrough şi IBM.

Conceput iniţial pentru a fi utilizat în regim conversaţional, în „timesharing" şi timp real şi asigurîndu-şi succesul în mare măsură tocmai datorită acestor calităţi, limbajul BASIC a fost apoi adaptat pentru a putea fi folosit şi în regim secvenţial.

Utilizarea BASIC-ului în modul secvenţial (batch mode) a fost realizată pentru prima oară la universitatea din Washington sub conducerea lui William F. Sharpe, [40].

Compilatorul care face posibilă utilizarea limbajului BASIC în „batchmode" (programele sînt perforate pe bandă de hîrtie sau pe cartele, în loc să fie introduse din claviatura unei console) poartă denumirea de UWBIC*. Acest procesor este scris în FORTRAN IV astfel încît poate fi implementat pe o gamă largă de calculatoare de talie mijlocie (în mod normal toate firmele producătoare de calculatoare de mărime mijlocie furnizează compilatoare FORTRAN IV).

Procesorul UWBIC lucrează în prezent pe un număr mare de calculatoare ce utilizează hardware IBM 360, model 50 şi 75, IBM 7094, IBM 7044/7094, Univac 1108 şi Control Data 3600 şi 6600.

Comparat cu modul de lucru conversaţional, modul secvenţial oferă un singur avantaj; programul şi datele se introduc automat de pe cartele sau bandă, care au fost perforate „of line" de către utilizator.

Se realizează un cîştig în ceea ce priveşte timpul de ocupare a sistemului pentru introducerea programelor şi a datelor. În schimb păstrează toate deza-

---

*** *U*niversity of *W*ashington *BAS*IC *I*nterpretiv *C*ompiller (Compilator interpretativ pentru BASIC al Universităţii din Washington).**

vantajele unui sistem de lucru secvențial, dezavantajele ce au fost expuse anterior.

Între limbajele „time-sharing BASIC" și „batch-mode BASIC" există o serie de diferențe datorate în primul rînd dispozitivelor de intrare/ieșire diferite pe care le utilizează. De exemplu simbolurile „>" sau „<" nu se află în dotarea claviaturilor unora din mașinile de perforat (IBM 026, ș.a.), ceea ce face ca simbolurile relaționale de comparare să fie codificate în „batch-mode BASIC" prin caractere alfabetice: LT-Less Then — (echivalentul lui <), LE-Less Then or Equale — (echivalentul lui < =), ș.a.m.d.

Pe de altă parte în „time-sharing BASIC" toate instrucțiunile trebuie să fie etichetate. Compilatorul le aranjează în secvență după ordinea etichetelor, indiferent de ordinea cronologică a introducerii din claviatură. Instrucțiunea END care marchează sfîrșitul programului trebuie să aibă întotdeauna eticheta cea mai mare.

În sistemul secvențial (batch-mode BASIC) nu e necesar ca toate instrucțiunile să fie etichetate, secvența instrucțiunilor fiind dată de ordinea de citire a cartelelor.

Deși există și alte diferențe între cele două forme ale limbajului, ele rămîn totuși neesențiale astfel încît un cunoscător al limbajului poate să-și adapteze cu ușurință programul de la modul conversațional la modul secvențial și invers.

Capitolul II

MINICALCULATORUL N O V A

2.1. STRUCTURĂ ŞI CARACTERISTICI

Printre primele calculatoare pe care s-a aplicat limbajul BASIC la noi în ţară au fost minicalculatoarele din seria NOVA, produse de către firma "Data General Corporation". Considerăm că este utilă o prezentare generală a structurii de „hardware" şi „software" a acestui minicalculator, prin faptul că el prezintă particularităţi pronunţate faţă de alte calculatoare, care au mai fost descrise în literatura noastră de specialitate. Această descriere generală va ajuta la o mai bună înţelegere a utilizării limbajului BASIC, deşi acesta nu este orientat pe maşină.

Calculatoarele din seria NOVA sînt sisteme de calcul cu cîmp larg de utilizare avînd organizarea pe cuvînt de 16 biţi. Toate calculatoarele din serie folosesc ca element esenţial în organizarea lor un grup de 4 registre acumulatoare, dintre care două pot fi folosite ca registre de index. Aceste acumulatoare oferă facilităţi în programare şi o mare eficienţă în utilizarea timpului şi a memoriei.

Diferenţele între modele în cadrul seriei NOVA apar în privinţa performanţelor şi a facilităţilor, dar toate folosesc acelaşi set de instrucţiuni şi programarea este absolut compatibilă, [61].

Stocarea, manipularea şi prelucrarea informaţiei se face la nivel de cuvînt de 16 biţi. Un cuvînt poate fi o instrucţiune de program, o adresă sau un operand. La rîndul său un operand poate fi interpretat de către program ca un cuvînt logic, o adresă, o pereche de doi octeţi sau ca un număr binar, cu sau fără semn format din 16 biţi.

Unitatea centrală a calculatorului NOVA execută programul prin derularea instrucţiunilor preluate din locaţii de memorie consecutive. Ordinea de executare a instrucţiunilor este dată de un registru contor de program (program counter). La sfîrşitul fiecărei instrucţiuni acesta îşi incrementează conţinutul cu o unitate astfel că noua instrucţiune este adusă dintr-o locaţie consecutivă. Desfăşurarea secvenţială a programului poate fi modificată prin schimbarea conţinutului contorului de program, fie printr-o incrementare suplimentară în urma unor testări, fie prin înlocuirea conţinutului cu o valoare specificată de către o instrucţiune de salt.

Alte registre interne de importanţă deosebită pentru programator sînt cele 4 registre acumulatoare: AC 0, AC 1, AC 2 şi AC 3. Datele pot fi transferate în orice direcţie între orice locaţie de memorie şi oricare din aceste patru acumulatoare.

Deşi un cuvînt din memorie poate fi incrementat sau decrementat, toate celelalte operaţii aritmetice sau logice se aplică numai pe operanzi ce rezidă în acumulatoare, iar rezultatul se plasează tot într-un acumulator. Prin urmare instrucţiunile aritmetice şi logice nu fac niciodată referiri la memorie. Memoria este folosită pentru a stoca programul şi datele permanente, iar pentru calcule şi rezultate intermediare sînt utilizate acumulatoarele. Aceasta micşorează considerabil timpul necesar manipulării datelor şi reduce numărul de instrucţiuni folosite în program.

Pentru exemplificare vom folosi o operaţie simplă constînd în schimbarea conţinutului a două locaţii de memorie, A şi B în două variante:

a) Cu utilizarea unui singur acumulator (AC)

A → AC
AC → ZONA
B → AC
AC → A
ZONA → AC
AC → B

b) Cu utilizarea a două acumulatoare (AC 1 şi AC 2)

A → AC 1
B → AC 2
AC 1 → B
AC 2 → A

ZONA este numele unei celule de memorie folosită ca auxiliar în acest exemplu.

Se remarcă faptul că în cazul (b) numărul de operaţii este cu 1/3 mai mic şi că nu mai este necesară utilizarea celulei ZONA (economie de memorie).

În figura 2.1 se prezintă o configuraţie tipică de sistem NOVA. În cadrul chenarului punctat este cuprinsă unitatea de bază (basic unit) formată din unitatea centrală de prelucrare, blocuri de memorie destructivă, pe miezuri magnetice sau circuite basculante bistabile integrate de tip MOS*, memorie de tipul „citeşte numai“ (ROM**) şi unităţi de control pentru diferite periferice.

Toate blocurile funcţionale enumerate mai sus se află încorporate fizic într-un singur dulap, purtînd denumirea de unitate de bază a configuraţiei. Figura 2.1 trebuie înţeleasă ca un exemplu de arhitectură NOVA şi nu ca o configuraţie standard imuabilă. În unele configuraţii, unitatea de bază cuprinde pe lîngă unităţi de control periferic, chiar şi dispozitive periferice. Ca exemplu cităm cazul minicalculatorului NOVA-840, aflat în dotarea laboratoarelor Catedrei de Cibernetică Economică din A. S. E. Bucureşti, care are încorporat în dulapul unităţii de bază un cititor/perforator de bandă şi un disc cu capete fixe.

Caracteristic pentru o structură de sistem NOVA este faptul că unitatea centrală — ca unitate de control a întregului sistem — este conectată la restul componentelor prin două linii magistrale; una care asigură legătura cu memoria (Memory Bus) şi alta care asigură legătura cu echipamentul periferic (Input/Output Bus).

Pe aceste linii informaţia circulă la nivel de cuvînt de 16 biţi sub controlul unităţii centrale.

La magistrala de intrare/ieşire se pot lega — din punct de vedere “hardware” — pînă la 62 dispozitive periferice.

---

* Metal—Oxid Semiconductor.
** Read Only Memory.

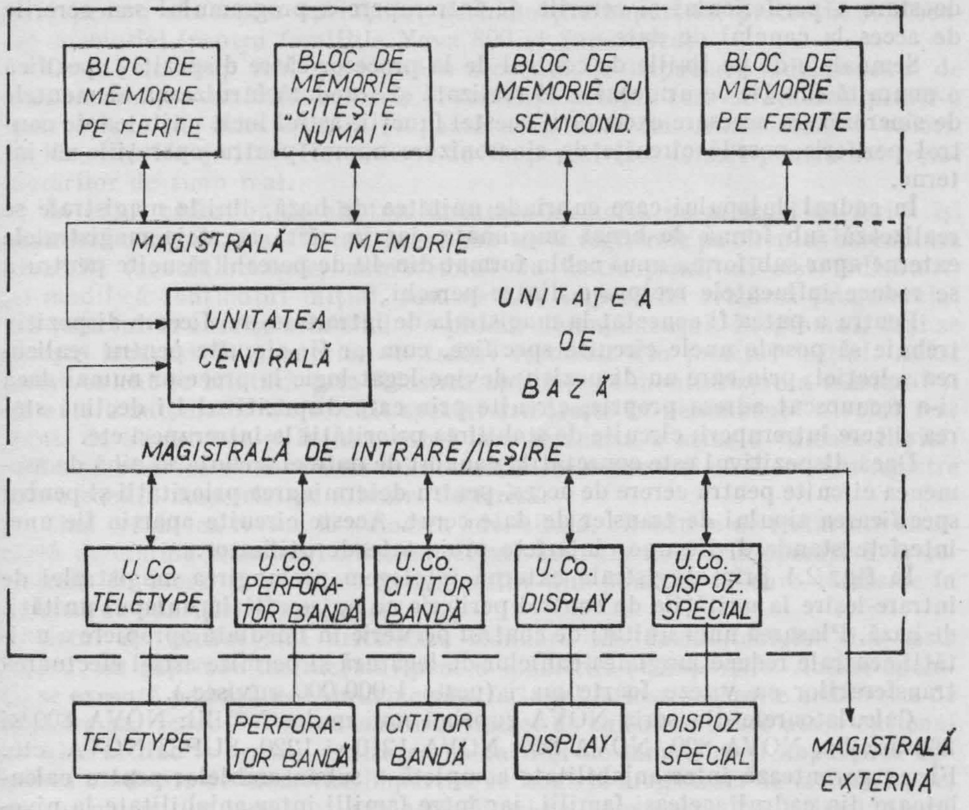


Fig. 2.1

Schimburile de date între periferie şi memorie se pot realiza în două moduri diferite:

a) sub controlul programului, caz în care cuvîntul transferat trece mai întîi printr-un acumulator şi abia apoi este dus din acumulator într-o locaţie de memorie. Acest mod de lucru necesită mai multe instrucţiuni pentru transferarea unui cuvînt, precum şi controlul permanent al unităţii centrale;

b) acces direct la memorie printr-un canal de date, cu o participare minimă a unităţii centrale. Acest mod de lucru este folosit de către dispozitivele periferice cu rate de transfer mare, ca: discuri magnetice, benzi magnetice, casete magnetice etc.

Magistrala de intrare/ieşire, este formată din 16 linii bidirecţionale (biţii unui cuvînt), denumită şi magistrală de date, 6 linii pentru selectarea perifericului, 19 linii de control de la unitatea centrală la dispozitivul periferic şi 6 linii de control în sens invers.

Semnalele care circulă pe liniile de control de la procesor* la dispozitiv sincronizează toate transferurile de pe magistrala de date, startează şi opreşte operaţiile la dispozitiv şi controlează întreruperile de program şi canalul de date. Pe liniile de control de la periferie la procesor se transmit informaţiile

* Pentru unitatea centrală se foloseşte şi termenul de „procesor".

de stare a perifericului şi cererile de întrerupere a programului sau cererile de acces la canalul de date.

Semnalele de pe liniile de control de la procesor către dispozitiv specifică o anumită funcţie ce urmează a fi realizată şi totodată furnizează elementele de sincronizare necesare executării acestei funcţii, astfel încît unitatea de control periferic posedă circuite de sincronizare numai pentru operaţiile ei interne.

În cadrul dulapului care cuprinde unitatea de bază, liniile magistrale se realizează sub formă de benzi imprimate, iar în afara acesteia magistralele externe apar sub forma unui cablu format din 40 de perechi răsucite pentru a se reduce influenţele reciproce dintre perechi.

Pentru a putea fi conectat la magistrala de intrare-ieşire, fiecare dispozitiv trebuie să posede unele circuite specifice, cum ar fi: circuite pentru realizarea selecţiei, prin care un dispozitiv devine legat logic la procesor numai dacă şi-a recunoscut adresa proprie, circuite prin care dispozitivul îşi declină starea şi cere întreruperi, circuite de stabilirea priorităţii la întreruperi etc.

Dacă dispozitivul este conectat la canalul de date el trebuie să aibă de asemenea circuite pentru cerere de acces, pentru determinarea priorităţii şi pentru specificarea tipului de transfer de date cerut. Aceste circuite aparţin fie unei interfeţe standard, fie unor interfeţe proiectate de utilizator.

În fig. 2.1 prin magistrală externă înţelegem prelungirea magistralei de intrare-ieşire la unităţile de control periferic ce nu se află în dulapul unităţii de bază. Plasarea unor unităţi de control periferic în imediata apropiere a unităţii centrale reduce lungimea cablelor de legătură şi permite astfel efectuarea transferurilor cu viteze foarte mari (peste 1 000 000 cuv/sec.).

Calculatoarele din seria NOVA cuprind mai multe familii: NOVA 800 şi 860 Jumbo, NOVA 820, NOVA 840, NOVA 1210 şi 1220, SUPERNOVA etc. Firma garantează interşanjabilitate completă a subansamblelor pentru calculatoare din cadrul aceleaşi familii, iar între familii interşanjabilitate la nivelul interfeţelor dispozitivelor de intrare/ieşire, [61].

Toate calculatoarele au aceeaşi proiectare fundamentală din punct de vedere mecanic, electric şi electronic, chiar dacă diferă în dimensiuni şi configuraţie internă. Diferenţele între familii se limitează în special la capacitatea fizică şi interşanjabilitatea între subansamble. Modulele de memorie se construiesc cu capacităţile de 1 K, 2 K, 4 K, 8 K cuvinte.

Calculatoarele din familiile Nova 800, 820, 840, au ciclul complet de memorie de 800 nsec şi execută instrucţiuni aritmetice şi logice într-un singur ciclu. Cele din familiile 1200, 1210, 1220 au ciclul complet de memorie de 1200 nsec. şi execută instrucţiuni aritmetice şi logice în 1350 nsec.

Memoria pe ferite a familiei Supernova are un ciclu de 800 nsec, dar această familie poate să lucreze de asemenea cu memorie pe semiconductori (circuite MOS), caz în care ciclul de memorie este de numai 300 nsec., timp în care se execută o instrucţiune aritmetică sau logică.

Seria de minicalculatoare „NOVA“ prezintă posibilitatea utilizării unor echipamente suplimentare denumite facilităţi sau opţiuni hardware*. Acestea sînt: ceas de timp real, monitor de putere, cu posibilitatea restartării automate a programului în momentul restabilirii electroalimentării (după cădere), dispo-

---

* Prin „hardware“ se înţelege partea de echipamente şi circuite a unui calculator, adică tot ce este realizat fizic. Termenul de limbă engleză s-a încetăţenit în vorbirea curentă datorită faptului că e greu traductibil.

zitiv pentru efectuarea operaţiilor de înmulţire/împărţire, protecţia şi alocarea memoriei (pentru familiile Nova 800 şi Supernova).

Ceasul de timp real generează o secvenţă de impulsuri independentă de baza de timp a unităţii centrale. Frecvenţa acestui ceas se fixează printr-o instrucţiune de intrare/ieşire şi poate avea una din următoarele 3 valori: 10 Hz, 100 Hz şi 1000 Hz. Ceasul de timp real este folosit pentru controlul lucrărilor de timp real.

În cazul căderilor de electroalimentare memoria pe miezuri de ferită îşi păstrează conţinutul nealterat, în schimb toate registrele pe circuite basculante bistabile (acumulatoarele, numărătorul de instrucţiuni, diverşi indicatori) îşi modifică conţinutul iniţial, ceea ce face imposibilă reluarea programului din punctul în care se afla cînd s-a produs deranjamentul. Programul, deşi se găseşte încărcat în memorie, va trebui restartat din punctul iniţial. Acest inconvenient se înlătură prin folosirea opţiunii de monitorizare a tensiunii şi restartare automată. Un circuit special urmăreşte în permanenţă nivelul tensiunii de alimentare şi în momentul în care acesta a atins o valoare limită, declanşează trecerea automată a informaţiei de control şi de lucru din registre într-o zonă a memoriei pe miezuri de ferită.

Cînd tensiunea revine la valoarea admisibilă, circuitele de restartare automată determină transferul informaţiei salvate, din memoria operativă în registrele de lucru şi pornirea programului din punctul în care rămăsese în momentul apariţiei deranjamentului.

Setul de instrucţiuni aritmetice standard ale calculatoarelor din seria „Nova" nu cuprinde instrucţiuni pentru înmulţire şi împărţire. Aceste operaţii se execută pe bază de subrutine care fac apel la instrucţiunile aritmetice de adunare sau scădere. Pentru mărirea vitezei de calcul se poate ataşa opţional un bloc aritmetic pentru realizarea operaţiilor de înmulţire şi împărţire. Opţiunea hardware de înmulţire/împărţire se leagă la magistrala de intrare/ieşire, deşi ea nu conţine indicatori sau posibilităţi de întreruperi. Înmulţirea durează 6,4 µsec. iar împărţirea între 6,8 şi 7,2 µsec. Deîmpărţitul şi rezultatul înmulţirii sînt întotdeauna operanzi de lungime dublă (două cuvinte Nova), ceilalţi operanzi sînt de lungime simplă.

În limbaj de asamblare se folosesc mnemonicele MUL (Multiply) pentru înmulţire şi DIV (Divide) pentru împărţire.

Facilitatea de alocare şi protecţie a memoriei permite prelucrarea aparent simultană a mai multor programe pe acelaşi calculator. Fără această facilitate sistemul de calcul execută un singur program care nu e supus la restricţii de utilizarea memoriei cu excepţia celor impuse de hardware; programatorul poate folosi întreaga capacitate de memorie disponibilă.

Seriile „Nova 800" şi „Supernova" pot fi dotate cu opţiunea de alocare şi protecţie a memoriei, fapt ce introduce restricţii în utilizarea calculatorului, dar permite divizarea timpului unităţii centrale între mai multe programe (time-sharing). În acest regim de lucru programele beneficiarilor sînt privite şi tratate de către procesor într-o manieră specială denumită „modul utilizator" Programul trebuie să opereze în cadrul unei zone de memorie prestabilite, iar folosirea unora dintre instrucţiuni este considerată ilegală în modul „utilizator".

Pentru coordonarea activităţii intreţesute a programelor de utilizator este prevăzut un program aparţinînd sistemului de operare — denumit program executiv — care planifică lansarea în execuţie a programelor de utilizator,

înregistrează şi prelucrează întreruperile, rezolvă cererile de intrare/ieşire şi în general planifică resursele fizice şi logice şi controlează întregul sistem de lucru bazat pe multiprogramare.

Fiecare utilizator are alocată o partiţie proprie de memorie şi nu poate avea acces în nici o altă zonă, nici pentru stocare, şi nici pentru regăsire de informaţii. Mai mult decît atît, chiar în cadrul propriei partiţii, la anumite zone fixate de către executiv, programul nu poate avea decît acces parţial, este permisă citirea dar interzisă scrierea. Aceasta se întîmplă în cazul procedurilor pure care pot fi folosite reentrant de către mai mulţi utilizatori.

Minicalculatoarele din seria „Nova" sînt dotate cu un software* puternic şi flexibil care cuprinde asambloare, editoare, compilatoare, sisteme de operare precum şi numeroase programe utilitare pentru diferite dispozitive, subrutine de depanare dinamică a programelor, convertoare de date, subrutine matematice şi interpretative şi un set complet de diagnosticare hardware.

Iată o trecere în revistă a principalelor componente software cu o descriere sumară a funcţiilor pe care le realizează, [62].

a) *Programele de asamblare „Absolute Assembler" şi „Extended Assembler"* sînt asambloare în doi paşi care produc coduri binare absolute. „Extended Assembler" realizează în plus relocarea, comunicaţia interprogram, asamblarea condiţionată şi facilităţi mai puternice de definirea numerelor.

b) *Asambloarele speciale „Cross Assemblers for IBM 360, CDC 6600, Univac 1108"*, toate scrise în Fortran IV, translatează programele scrise în limbaj sursă simbolic în cod-maşină Nova, folosind intrarea pe cartele perforate către IBM 360, CDC 6600 sau UNIVAC 1108. Ieşirea rezultă în coduri binare absolute sau relocatabile, acceptate ca intrare pentru încărcătorul binar.

c) *Programul de încărcare „Relocatable Binary Loader"* realizează încărcarea de pe benzi a programelor scrise în forma binară relocatabilă, produse de către Extended Assembler.

d) *Interpreterul „Time Sharing BASIC"* este un sistem interpretativ care acceptă intrarea conversaţională şi executarea programelor scrise în limbaj BASIC. El permite folosirea tuturor instrucţiunilor BASIC elementare şi complexe, inclusiv cele destinate lucrului cu matrice şi variabile de tip şir. Sistemul poate deservi în time-sharing pînă la 16 terminale teletype** şi include o listă comprehensivă de mesaje de erori.

e) *Interpreterul „Single User BASIC"* are toate facilităţile lui „Time Sharing BASIC" cu excepţia funcţiilor de manipulare a matricilor şi a variabilelor de tip şir.

f) *Interpreterul „Extended BASIC"* are faţă de „Time Sharing BASIC" facilităţi suplimentare în ceea ce priveşte accesul la fişierele de date sau programe de pe dispozitivele periferice de intrare/ieşire. Dispozitivele suport de fişiere includ cititoare şi perforatoare de viteză mare, imprimantă, disc cu capete fixe, disc cu capete mobile etc. Fişierele pe disc pot fi protejate contra accesului neautorizat sau pot fi plasate într-un sistem de bibliotecă, accesibil tuturor utilizatorilor. Versiunile pe disc ale compilatorului „Extended BA-

* Prin software se înţelege totalitatea programelor de control şi utilizare, furnizate de firmă şi care împreună cu hardwarul realizează prelucrarea automată a programelor de utilizator. Întrucît termenul de software ca şi cel de hardware este greu traductibil va fi folosit în continuare ca atare.

** Teletype este un periferic de intrare/ieşire cu claviatură (maşină de scris) utilizat de la distanţă.

ISC" împart memoria internă între utilizatori, în sistem „time-sharing", permițînd fiecărui utilizator accesul la toată memoria.

g) *Compilatorul „Extended FORTRAN IV"* este o implementare a lui ANSI FORTRAN IV, [64], cu posibilități de utilizare reentrantă în cod obiect și cu multe extensii de limbaj, cum ar fi: expresii cu indici generalizați, aritmetică în complex dublă precizie, utilizarea de vectori cu 128 de componente, declararea unor vectori la care marginea inferioară nu trebuie să fie 1, ș.a.

h) *Compilatorul „Extended ALGOL"* este o versiune îmbunătățită a lui ALGOL 60 cu extensii care permit simplificări în operațiile de I/E, manipularea la nivel de bit facilități în manipularea șirurilor de caractere, utilizarea procedurilor reentrante și recursive, alocarea dinamică a memoriei, lucrul cu vectori n-dimensionali, aritmetică multiprecizie, diagnosticare explicită, ș.a.

i) *Sistemul de operare „Real Time Disc Operating System (RDOS)*, este un sistem de operare modular, în timp real, parțial rezident în memoria operativă și parțial rezident pe disc.

RDOS, ca sistem de operare în timp real poate planifica și aloca controlul programului la mai multe sarcini de program (multi-task) diferite, pentru a realiza utilizarea simultană a resurselor sistemului, fapt ce mărește randamentul prelucrării și asigură eficiență și economie în exploatarea resurselor.

Sistemul de operare RDOS poate fi utilizat interactiv de la claviatură de consolă sau secvențial (Batch mode) prin lucrări (job streams) lansate de la un cititor de cartele. Faptul că o parte a sistemului de operare este plasat parțial pe disc mărește spațiul de memorie operativă disponibil utilizatorilor și oferă în același timp un spațiu de memorie mai mare chiar pentru sistem, lucru ce permite dotarea cu o serie întreagă de funcții suplimentare și facilități a acestuia.

j) *Sistemul de operare „Real Time Operating System" (RTOS)* este de asemenea un sistem modular în timp real, cu prelucrare aparent simultană a sarcinilor de program, dar care rezidă integral în memoria operativă și are dimensiuni și facilități mai reduse decît RDOS. Asistate de către RTOS, programele de utilizator sînt eliberate de problemele de sincronizare a operațiilor de intrare/ieșire, manipularea datelor, evidența priorităților și planificarea sarcinilor de program. În plus sarcinile sînt prevăzute cu posibilități de prelucrare paralelă și facilități de comunicare și sincronizare între ele.

Se consideră că RTOS este un subset compatibil al lui RDOS.

k) *Sistemul de operare „Disc Operating System" (DOS)* este un sistem de operare clasic, parțial rezident pe disc. Prevede facilități de lucru cu fișiere, administrarea memoriilor tampon, gestiunea și supravegherea operațiilor de I/E, gestiunea întreruperilor, facilități de comunicare cu programul și cu operatorul, etc.

2.2. TIPURI DE INSTRUCȚIUNI ȘI MODURI DE ADRESARE

În funcție de operațiile pe care le realizează, setul de instrucțiuni al calculatoarelor din seria NOVA se împarte în 5 clase:

1. *Instrucțiuni privind transfer de date de la memorie la registrele acumulatoare sau invers.* Participă o locație de memorie și un acumulator. Fiecare dintre ele poate fi sursă sau destinație a operandului.

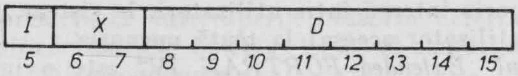


Fig. 2.2

2. *Instrucţiuni de modificare a memoriei*. Implică o singură locaţie de memorie căreia i se incrementează sau decrementează conţinutul. Dacă se ajunge la conţinut zero are loc un salt peste următoarea instrucţiune de program.
3. *Instrucţiuni de salt*. Implică de asemenea o singură locaţie de memorie de la care se ia noua instrucţiune. Adresa de reîntoarcere în program poate fi salvată în acumulatorul AC 3.

4. *Instrucţiuni aritmetice şi logice*. Participă două acumulatoare, al căror conţinut este implicat în operaţii aritmetice sau logice. Pe lîngă operaţiile aritmetice sau logice propriu zise se mai efectuează, prin aceeaşi instrucţiune şi alte operaţii suplimentare cum ar fi deplasarea ciclică cu o poziţie a rezultatului, schimbarea reciprocă de poziţii între cele două jumătăţi ale cuvîntului rezultat, analiza şi poziţionarea bitului de transport în funcţie de anumiţi biţi indicatori din codul instrucţiunii, etc.

5. *Instrucţiuni de intrare/ieşire*. Participă un acumulator şi un dispozitiv de intrare/ieşire. Ca urmare a executării unei instrucţiuni de I/E are loc transferul unui cuvînt în orice direcţie între un acumulator şi unul din trei registre ale unui dispozitiv periferic.

În instrucţiune se specifică tipul operaţiei, tipul dispozitivului, registrul acumulator şi registrul dispozitivului.

Instrucţiunile din primele trei clase trebuie să adreseze o locaţie de memorie. Din punct de vedere al adresării întreaga memorie este privită ca o mulţime de locaţii contigue ale căror adrese încep de la zero şi merg pînă la o valoare maximă depinzînd de capacitatea fizică instalată. Memoria este alcătuită din module avînd capacităţi de 1024, 2048, 4096 sau 8192 cuvinte pentru memoria destructivă şi 256, 512 sau 1024 cuvinte pentru memoria nedestructivă (ROM).

O adresă furnizată de către program este decodificată în două părţi; partea cea mai semnificativă care selectează modulul de memorie şi partea mai puţin semnificativă care selectează cuvîntul în cadrul modulului, dar această organizare este transparentă pentru programator.

Toate instrucţiunile cu referire la memorie au acelaşi format în cîmpul ce ocupă biţii 5—15 (fig. 2.2).

Fiecare din cele 3 zone aparţinînd acestui cîmp realizează funcţii specifice în cadrul adresării. Bitul 5 specifică tipul adresării — directă sau indirectă —, biţii 6 şi 7 sînt biţi de index, iar biţii 8—15 reprezintă deplasarea. Adresa efectivă* utilizată de instrucţiune depinde de valorile lui I, X şi D.

Dacă X=00, cîmpul D adresează una din primele 256 locaţii de memorie, iar operaţia poartă denumirea de adresare în pagina zero.

Dacă X≠00, D reprezintă o deplasare care se adună la conţinutul registrului acumulator specificat de X, pentru a produce o adresă de memorie.

---

* Prin adresă efectivă înţelegem o adresă la care se depune sau de la care se extrage un operand.

Deplasarea D este un întreg binar cu semn exprimat în cod complementar (complementul față de 2), luînd valori în gama (−128) ... (+127), sau în exprimare octală* de la $(-200)_8$ ... $(+177)_8$.

Pentru X=01, cîmpul D se adună la conținutul contorului de program pentru a furniza o adresă de memorie. Acest mod de adresare se numește adresare relativă și oferă oricărei instrucțiuni de program posibilitatea de a se referi la 255 de locații din propria-i vecinătate (deplasarea ia valori de la −128 la +127).

Pentru X=10 sau X=11, adresa de memorie specificată prin instrucțiune se obține adunînd deplasarea la conținutul registrului acumulator AC 2, respectiv AC 3.

Dacă I=0, sîntem în cazul adresării directe și adresa determinată din X și D, într-unul din modurile expuse mai sus, este o adresă efectivă folosită în executarea instrucțiunii.

Dacă I=1, adresarea se numește indirectă și în acest caz adresa obținută din X și D este adresa unei locații de memorie care conține o nouă adresă, deja calculată și înscrisă în biții 1−15. Dacă bitul 0 al noii locații este 0, înseamnă că adresa înscrisă în pozițiile 1−15 este o adresă efectivă, adică adresa unui operand. Dacă bitul 0 este egal cu 1 se trece la un nou nivel de adresare indirectă.

Spațiul de adrese este ciclic în raport cu operațiile efectuate pentru calculul unei adrese efective; indiferent de valoarea obținută pentru sumă sau diferența într-o etapă oarecare de calcul, numai ultimele 15 poziții (mai puțin semnificative) din cuvînt sînt utilizate ca adresă. Prin urmare adresa care urmează după $(77777)_8$ este $(00000)_8$.

Unele locații sînt prevăzute cu o facilitate hardware de incrementare și decrementare automată a conținutului. Dacă la orice nivel de calcul a unei adrese efective, un cuvînt de adresă este adus dintr-una din locațiile $(00020)_8$ ... ... $(0037)_8$ el este incrementat sau decrementat automat cu o unitate, modificare ce se operează și în memorie. Adresele luate din locațiile $(00020)_8$ ... ... $(00027)_8$ sînt incrementate, iar cele luate din locațiile $(00027)_8$ ... $(00037)_8$ sînt decrementate.

Pasul următor în calcularea adresei efective depinde de biții 1−15 ai noului cuvînt de adresă și de bitul 0 al cuvîntului de adresă anterior. Dacă bitul 0 al cuvîntului adus inițial din memorie avea valoarea 0, noua valoare de adresă obținută prin incrementare sau decrementare va fi tratată ca o adresă efectivă. În caz contrar noua valoare de adresă va fi folosită pentru a aduce un nou cuvînt în calculul valorii adresei efective.

Să analizăm acum caracteristicile fiecărei clase de instrucțiuni.

1. Instrucțiuni de transfer între memorie și acumulatoare. Formatul general al instrucțiunii este cel din figura 2.3.

0	01 / 10	ADR AC	I	X	D
0	1 2	3 4	5	6 7	8 ... 15

Fig. 2.3

* Pentru calculatoarele din seria NOVA este foarte frecventă exprimarea în octal a conținutului unor registre sau locații de memorie.

Bitul 0 este întotdeauna egal cu 0 şi specifică instrucţiune nearitmetică. Biţii 1 şi 2 specifică funcţia şi au următoarea semnificaţie:

a) $B_1B_2=01$, transferul are loc de la o locaţie de memorie a cărei adresă efectivă se calculează din cîmpurile I, X şi D, într-un acumulator a cărui adresă este specificată de biţii 3, 4.

În limbaj assembler această funcţie este reprezentată prin monemonicele LDA (LoaD Accumulator) care în traducere înseamnă „Încărcare acumulator".

b) $B_1B_2=10$, transferul are loc dintr-un acumulator a cărui adresă este specificată de biţii 3, 4, într-o locaţie de memorie a cărei adresă efectivă se calculează din cîmpul biţilor 5—15.

În assembler funcţia se exprimă prin STA (STore Accumulator) care înseamnă „Stochează în memorie conţinutul acumulatorului".

Exemple de instrucţiuni scrise în limbaj assembler:

a) LDA 3, -34, 2. După asamblare cuvîntul instrucţiune va avea conţinutul: 035344_8. Instrucţiunea exprimă următoarele: încarcă în acumulatorul 3 conţinutul locaţiei a cărui adresă efectivă se obţine din $D=-34$, $X=2$ şi $I=0$. Fără nici o specificare, adresarea se presupune directă, adică $I=0$. Cînd vrem să specificăm adresare indirectă este necesar ca D să fie precedat de semnul @. Exemplu:

LDA 3, @ —34, 2

Instrucţiunea este asemănătoare cu cea precedentă, cu deosebirea că adresa calculată din D şi X nu este o adresă efectivă ci adresa unei locaţii de memorie la care, eventual se află adresa efectivă. Alte exemple:

LDA 1, @ 120	Adresare indirectă în pagina zero.
LDA 2, @ —70,1	Adresare indirectă şi relativă (prin intermediul contorului de program).
LDA 2, 80	Adresare directă în pagina zero
LDA 2, —50, 3	Adresare directă prin registru de bază (AC 3).

b) STA 0, @—65, 2 exprimă: stochează conţinutul acumulatorului 0 la locaţia de memorie rezultată dintr-o adresare indirectă, indexată, cu indexul plasat în acumulatorul 2.

2. Instrucţiuni de salt şi de modificarea memoriei. Formatul general al acestei instrucţiuni este cel din figura 2.4. Biţii 0, 1, 2 au întotdeauna valoarea zero, biţii 3, 4, specifică funcţia, iar biţii 5—15 servesc pentru calculul adresei unei locaţii de memorie.

Iată descrierea celor 4 funcţii ce se realizează în cadrul acestei clase de instrucţiuni:

1) $B_3B_4=00$. Instrucţiune de salt (jump) reprezentată în assembler prin mnemonicul JMP. Operaţia realizată de această instrucţiune constă în trecerea în contorul de program a conţinutului locaţiei cu adresa efectivă calculată din I, X, D. În continuare programul va urma ordinea secvenţială pornind de la noul conţinut al contorului de program.

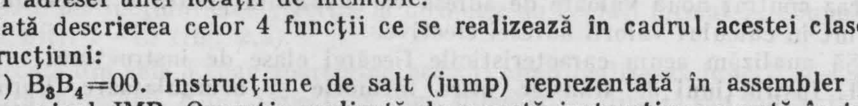


Fig. 2.4

2) $B_3B_4=01$. Instrucţiune de salt la o subrutină, reprezentată mnemonic prin JSR (Jump to Subroutine). Executarea instrucţiunii are ca efect transferarea conţinutului contorului de program — după ce a fost incrementat cu o unitate — în acumulatorul AC 3 şi plasarea în contorul de program a conţinutului locaţiei cu adresa efectivă rezultată din I, X, D. În continuare programul parcurge în ordine secvenţială o anumită subrutină, începînd cu noul conţinut al contorului şi iese din subrutină printr-o instrucţiune de forma JMP 0, 3, deoarece reîntoarcerea la vechea ordine secvenţială va fi dată de AC 3. De menţionat că adresa efectivă este deja calculată în momentul în care are loc transferarea conţinutului contorului de program în acumulatorul AC 3, aşa că AC 3 poate fi folosit ca registru de index în cadrul instrucţiunii JSR.

3) $B_3B_4=10$. Reprezentare mnemonică: ISZ (Increment and Spki if Zero). Se adună 1 la conţinutul locaţiei cu adresa efectivă rezultată din I, X, D şi dacă rezultatul adunării este zero se sare peste următoarea instrucţiune din program.

4) $B_3B_4=11$. Reprezentare mnemonică: DSZ (Decrement and Skip if Zero). Se scade 1 din conţinutul locaţiei cu adresa efectivă calculată din I, X, D şi dacă rezultatul este zero se sare peste următoarea instrucţiune de program.

3. Instrucţiuni aritmetice şi logice. În cazul operaţiilor logice calculatorul interpretează operanzii ca pe nişte cuvinte logice. În cazul operaţiilor aritmetice operanzii sînt trataţi ca numere fără semn formate din 16 ranguri binare, deci cuprinse în gama $0 \dots 2^{16}-1$. Programul interpretează totuşi operanzii aritmetici ca pe nişte numere cu semn, fapt pentru care s-a făcut apel la exprimarea în cod complementar. Presupunem că un acumulator are următorul conţinut:

1 000000001011001

Dacă acest conţinut se consideră un număr fără semn, valoarea sa va fi $100131_8=32857_{10}$; dacă se consideră un număr cu semn valoarea acestuia va fi $-77647_8=-32679_{10}$.

Deşi unitatea aritmetică propriu zisă nu face distincţie între numerele cu semn şi cele fără semn, tratîndu-le pe toate ca pe un şir de biţi, programul va interpreta întotdeauna bitul cel mai semnificativ (bitul zero) ca poziţie de semn şi va lua în consideraţie valoarea algebrică a numerelor, în convenţia codului complementar.

Pe de altă parte, operanzii aritmetici nu sînt întotdeauna numere întregi, putînd apare şi sub formă de fracţii zecimale. Calculatorul nu ia în consideraţie poziţia virgulei în momentul efectuării operaţiei aritmetice. Este sarcina programatorului să stabilească o anumită convenţie privind plasarea virgulei şi să scaleze operanzii şi rezultatele în funcţie de convenţia adoptată.

În general cele mai utilizate convenţii sînt: reprezentarea operaţiilor ca numere întregi (virgula plasată după poziţia cea mai puţin semnificativă) sau ca fracţii subunitare (virgula plasată după poziţia bitului de semn, adică imediat după cifra cea mai semnificativă). Conform celor două convenţii enunţate mai sus, gama de reprezentare a numerelor negative va fi $(-2^{15}) \dots$ $\dots (2^{15}-1)$ şi respectiv $(-1) \dots (1-2^{-15})$.

Deoarece fiecare poziţie de bit dintr-un număr reprezintă un ordin de mărime binară, deplasarea conţinutului unui registru cu o poziţie către stînga

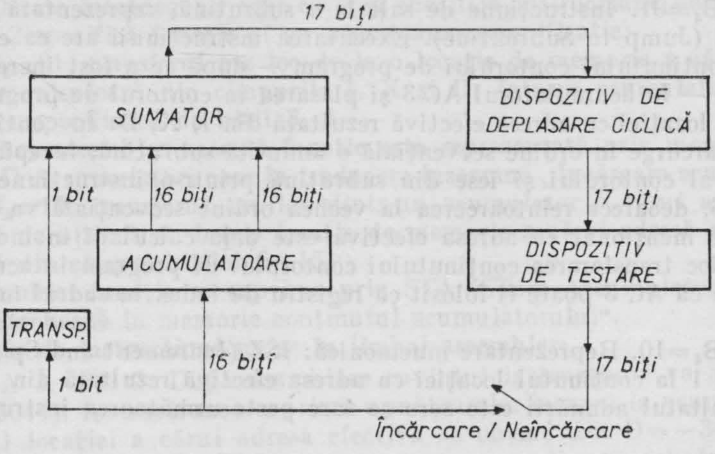


Fig. 2.5

echivalează cu înmulţirea numărului cu 2, presupunînd bineînţeles că virgula binară a rămas pe loc şi că nu s-a pierdut o cifră semnificativă. În mod similar, deplasarea către dreapta cu o poziţie a conţinutului unui registru, echivalează cu împărţirea numărului respectiv prin 2.

Aceste proprietăţi sînt folosite în cadrul operaţiilor de înmulţire şi împărţire.

Pentru o mai bună înţelegere a funcţiilor realizate de către instrucţiunile aritmetice şi logice este bine să analizăm mai întîi organizarea şi funcţionarea unităţii aritmetice (fig. 2.5).

Fiecare instrucţiune indică unul sau două acumulatoare ce furnizează operanzi către sumator, unde se efectuează funcţia specificată de către instrucţiune. Sumatorul produce de asemenea şi un bit de transport a cărui valoare depinde de trei elemente: o valoare iniţială specificată prin instrucţiune, funcţia efectuată şi valoarea efectiv obţinută pentru transport în urma executării funcţiei.

Din sumator iese întotdeauna un cuvînt format din 17 biţi: 16 biţi reprezentînd rezultatul operaţiei şi 1 bit de transport (depăşire). Acest cuvînt intră în continuare în dispozitivul de deplasare unde rezultatul — inclusiv transportul — poate fi deplasat circular cu o poziţie la stînga sau la dreapta. În acest dispozitiv se poate realiza de asemenea şi o inversare de poziţii între cele două jumătăţi ale rezultatului (fără bitul de transport), aşa cum se vede în figura 2.6.

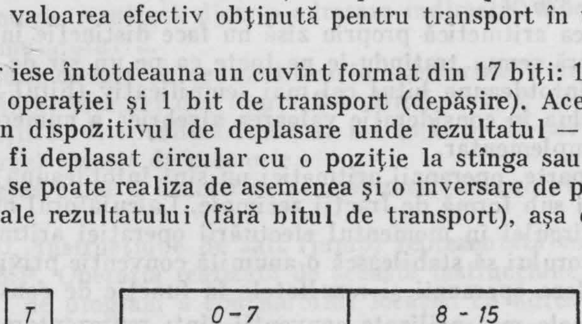


Fig. 2.6

1	ADR.AC. SURSĂ	ADR.AC. DESTIN.	FUNCTIA	DEPL.	TRANSP.	NE-INC.	SKIP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

Fig. 2.7

În continuare rezultatul operației intră într-un dispozitiv de testare prin care se urmărește detectarea unei condiții de skip*.

La ieșirea din acest dispozitiv cuvîntul poate fi încărcat într-unul din acumulatoare și în indicatorul de transport. Nu toate rezultatele operațiilor aritmetice sau logice se păstrează; necesitatea încărcării rezultatului în acumulator se specifică printr-un bit indicator în cadrul instrucțiunii.

Formatul general al instrucțiunilor aritmetice și logice este cel din figura 2.7. Prezența lui 1 în poziția 0 marchează clasa de instrucțiuni aritmetice. Funcția, înscrisă în pozițiile 5—7, care poate fi variată, începînd cu o simplă deplasare și terminînd cu o scădere, folosește conținutul acumulatorului specificat de pozițiile 1, 2, iar dacă este necesar și un al doilea operand, acesta se ia dintr-un alt acumulator, specificat prin pozițiile 3, 4.

Primul acumulator se numește „sursă" iar cel de al doilea „destinație" și este folosit și pentru plasarea rezultatului, cînd este cazul.

Instrucțiunea furnizează către dispozitivul de deplasare, împreună cu rezultatul și un bit de transport a cărei valoare depinde direct de biții 10 și 11 din instrucțiune. Cînd se scrie instrucțiunea în limbaj assembler, conținutul acestor biți este fixat printr-un mnemonic. Asignările de mnemonice și semnificațiile lor sînt cuprinse în tabelul 2.1.

Tabelul 2.1

Biții 10—11	Mnemonic	Valoarea atribuită bițului de transport
0 0		Valoarea curentă rezultată din sumator
0 1	Z	Zero
1 0	O	Unu
1 1	C	Complementul valorii curente rezultate din sumator

Funcțiile de deplasare circulară sînt specificate prin biții 8 și 9 din instrucțiune, iar explicitarea lor este dată în tabelul 2.2.

Tabelul 2.2

Biții 8—9	Mnemonic	Funcția realizată
0 0		Nici o funcție
0 1	L	Deplasare ciclică la stînga cu 1 poziție
1 0	R	Deplasare ciclică la dreapta cu 1 poziție
1 1	S	Inversarea jumătăților de cuvînt

* Prin skip, în acest capitol, înțelegem salt peste instrucțiunea următoare de program

Bitul 12 reprezintă un comutator de încărcare/neîncărcare a rezultatului în acumulator. Cînd are valoare 0 specifică încărcare, iar cînd are valoarea 1 specifică neîncărcare. În sfîrşit, ultimele trei poziţii (13, 14, 15) din instrucţiune specifică condiţiile de skip, condiţii ce sînt verificate în dispozitivul de testare.

Codificarea acestor condiţii şi reprezentările lor mnemonice în assembler sînt date în tabelul 2.3.

Tabelul 2.3

Biţii 13, 14, 15	Mnemonic	Condiţia de skip
0 0 0		Niciodată skip
0 0 1	SKP	Întotdeauna skip
0 1 0	SZC	Skip dacă transportul este zero
0 1 1	SNC	Skip dacă transportul este unu
1 0 0	SZR	Skip dacă rezultatul este zero
1 0 1	SNR	Skip dacă rezultatul este diferit de zero
1 1 0	SEZ	Skip dacă transportul sau rezultatul este zero
1 1 1	SBN	Skip dacă atît transportul cît şi rezultatul sînt diferite de zero

O instrucţiune aritmetică sau logică poate realiza, după cum s-a văzut, pe lîngă funcţia propriu zisă şi o serie de operaţii suplimentare, ca deplasarea circulară, inversarea jumătăţilor unui rezultat (swapping), testare la skip şi stabilirea valorii transportului.

În limbaj assembler, aceste funcţii suplimentare sînt specificate în cadrul instrucţiunii prin mnemonice. Formatul instrucţiunii va respecta următoarea sintaxă, cîmpurile fiind enunţate de la stînga către dreapta: codul operaţiei (funcţia), mnemonice legate de transport sau de deplasare, condiţia de neîncărcare, adresele acumulatoarelor (separate prin virgule) şi mnemonicele condiţiilor de testare. Pentru a ilustra cele de mai sus vom da mai multe exemple însoţite de explicaţiile corespunzătoare.

a) ADD 1, 2. Adună conţinutul acumulatoarelor 1 şi 2 şi plasează rezultatul nedeplasat în acumulatorul 2.

b) ADD ZL 1, 2. Adună conţinutul acumulatoarelor 1 şi 2, plasează în 2 rezultatul deplasat circular la stînga şi atribuie transportului valoarea zero.

c) ADD L 1, 2, SZC. Adună conţinutul acumulatoarelor 1 şi 2, plasează rezultatul deplasat circular în 2, atribuie transportului valoarea curentă şi efectuează skip dacă valoarea transportului este zero.

d) ADD L # 1, 2, SZR. Adună conţinutul acumulatoarelor 1 şi 2, deplasează rezultatul circular către stînga şi efectuează skip dacă valoarea rezultatului este zero. Simbolul # specifică faptul că rezultatul obţinut nu se conservă. El serveşte numai pentru testare la condiţia de skip.

Întrucît cîmpul afectat funcţiei în cadrul instrucţiunii este de 3 biţi, înseamnă că vor exista 8 funcţii distincte.

Să urmărim o descriere sumară a cîtorva dintre ele.

a) COM Complement/Complementare logică

$$B_5B_6B_7=0\,0\,0$$

Se plasează complementul logic al cuvîntului adus din acumulatorul sursă împreună cu bitul de transport atribuit, în dispozitivul de deplasare. Se efectuează operaţia de deplasare specificată de biţii 8 şi 9, iar ieşirea din dispozitiv se încarcă în acumulatorul destinaţie, dacă bitul 12 nu este egal cu 1. Se sare peste instrucţiunea următoare dacă ieşirea din dispozitivul de deplasare satisface condiţia specifică prin biţii 13, 14, 15.

b) INC Increment/Incrementare

$$B_5B_6B_7=0\,1\,1$$

Se adună 1 la conţinutul acumulatorului sursă şi se plasează rezultatul în dispozitivul de deplasare. Dacă această sursă conţine $2^{16}-1$, în stabilirea valorii bitului de transport se ia complementul cîmpului format din biţii 10—11; în caz contrar se ia valoarea specificată de cîmpul respectiv. În continuare se efectuează operaţia de deplasare specificată de biţii 8—9 şi dacă valoarea bitului 12 nu este 1 se încarcă ieşirea din dispozitivul de deplasare în acumulatorul destinaţie şi în registrul de transport. Se face salt peste instrucţiunea următoare din program dacă este satisfăcută condiţia specificată în cîmpul biţilor 13 ... 15.

c) ADD Add/Adunare

$$B_5B_6B_7=1\,1\,0$$

Se adună conţinutul acumulatorului sursă cu conţinutul acumulatorului destinaţie şi se plasează rezultatul în dispozitivul de deplasare. Dacă suma fără semn este $\geqslant 2^{16}$, pentru stabilirea bitului de transport se ia complementul valorii specificate de către cîmpul bitilor 10 şi 11; în caz contrar se ia valoarea specificată de acest cîmp. În continuare se efectuează operaţia de deplasare indicată de biţii 8 şi 9. Dacă bitul 12 este diferit de 1, se încarcă ieşirea din dispozitivul de deplasare în acumulatorul destinaţie şi în registrul de transport. Se efectuează salt peste următoarea instrucţiune de program dacă a fost satisfăcută condiţia specificată de biţii 13 ... 15.

4. Instrucţiuni de intrare/ieşire. Instrucţiunile de intrare/ieşire realizează transferul de date între unitatea centrală şi echipamentul periferic. O instrucţiune din această clasă este recunoscută prin formaţia de biţi 0 1 1 în poziţiile 0 ... 2.

Formatul general al instrucţiunilor de intrare/ieşire este cel din figura 2.8.

0	1	1	ADRESĂ ACUM.		FUNCŢIA TRANSFER			FUNCŢIA CONTROL		ADRESĂ DISP. PERIFERIC					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2.8

Biţii 10 ... 15 conţin adresa dispozitivului periferic la (de la) care se face transferul de date. Cu aceşti biţi se pot forma 64 de coduri distincte, dintre care se folosesc pentru adresarea perifericelor numai 62. Codul $0\,0_8$ nu este utilizat iar codul 77_8 e folosit pentru diverse funcţii speciale cum ar fi citirea cheilor de pe panoul frontal, controlul asupra întreruperilor de program, etc.

Fiecare dispozitiv este dotat cu circuite de selecţie (decodificare) proprii, cu ajutorul cărora îşi recunoaşte adresa plasată pe magistrala de intrare/ieşire.

Deasemenea, dispozitivul mai are şi trei registre indicatoare cu capacitatea de 1 bit, dintre care două participă direct la efectuarea operaţiei de transfer. Aceştia sînt indicatorii „Ocupat" (Busy) şi „Disponibil" (Done) care definesc împreună starea dispozitivului. Cînd ambii au valoarea zero dispozitivul periferic este inactiv. Trecerea sa în activitate se face prin instrucţiunea de intrare/ieşire, care în acelaşi timp înscrie 1 în registrul „Ocupat" pentru a indica starea de ocupat a dispozitivului. După fiecare unitate elementară de informaţie (caracter sau cuvînt) transferată, perifericul înscrie 0 în registrul „Ocupat" şi 1 în registrul „Disponibil". Prin aceasta el înştiinţează unitatea centrală că este gata să primească sau să expedieze un nou cuvînt sau caracter. Urmează o nouă instrucţiune de intrare/ieşire care înscrie 1 în registrul „Ocupat" şi 0 în registrul „Disponibil", ş.a.m.d.

Indicatorii „Ocupat" şi „Disponibil" se pot afla într-una din două stări distincte marcate prin 0 şi 1. Situaţia la un moment dat a celor doi indicatori, precum şi tranziţia lor de stare, defineşte o anumită etapă în realizarea procesului de intrare/ieşire. Acest lucru se poate urmări din tabelul 2.4.

Tabelul 2.4

Starea	Ocupat	Disponibil	Tranziţii de stări
Periferic inactiv	0	0	A ↓ ; D ↑
Periferic ocupat	1	0	B ↓ ; C ↑ ; D ↑
Periferic disponibil	0	1	

Legenda tranziţiilor de stări:

A — Startarea dispozitivului

B — Operaţie terminată

C — Reluarea procesului de transfer de date

D — Aducerea dispozitivului în stare inactivă

În tabelul 2.4 singura tranziţie de stare efectuată de către periferic este trecerea din „Ocupat" în „Disponibil". Celelalte se fac sub comanda instrucţiunii, mai precis prin interpretarea biţilor 8 şi 9 din instrucţiune, denumiţi biţi de control (fig. 2.8). Funcţiile realizate de aceşti biţi sînt sintetizate în tabelul 2.5.

Biţii 5 ... 7 din instrucţiune indică funcţia de transfer ce urmează a se executa iar biţii 3, 4 specifică unul din cele 4 acumulatoare ale unităţii centrale din care se extrage sau în care se depune cuvîntul transferat.

Tabelul 2.5

Biţii 8,9	Mnemonic	Funcţia realizată
0 0		Nici o acţiune
0 1	S	Startează dispozitivul înscriind 1 în „Ocupat" şi O în „Disponibil"
1 0	C	Înscrie O atît în „Ocupat" cît şi în „Disponibil", deci aduce dispozitivul în stare inactivă
1 1	P	Generează un semnal pe linia de control a magistralei de intrare/ieşire. Efectul depinde de dispozitiv

Unele instrucţiuni de intrare/ieşire nu prevăd transferuri de date ci numai exercitarea unor funcţii de control la dispozitiv. În acest caz biţii 5, 6, 7 sînt toţi 0 sau toţi 1, biţii 3, 4 se ignoră, iar biţii 8, 9 specifică o funcţie de control sau o condiţie de skip.

Să urmărim în continuare cîteva exemple de instrucţiuni de intrare/ieşire.

a) NIO No Input-Output Transfer (Nici un transfer la intrare-ieşire). Formatul instrucţiunii este cel din figura 2.9. Se efectuează funcţia de control specificată de F asupra dispozitivului cu adresa D.

b) SKPBZ Skip if Busy is Zero (Salt dacă registrul „Ocupat" este zero). Formatul instrucţiunii poate fi urmărit în figura 2.10.

Se testează indicatorul de „Ocupat" din dispozitivul a cărui adresă este dată de D. Dacă se găseşte zero se face salt peste următoarea instrucţiune din program.

Instrucţiunile prezentate anterior fac parte din categoria instrucţiunilor de intrare/ieşire ce nu comportă transfer de date. Să urmărim şi exemple de instrucţiuni prin care se realizează operaţii de transfer.

c) DIA Data In A (Operaţie de intrare din registrul tampon A). În figura 2.11 se prezintă formatul acestei instrucţiuni, care realizează transferul

0	1	1	0	0	0	0	0	0	1	D					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2.9

0	1	1	0	0	0	0	0	F		D					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2.10

0	1	1	AC		0	0	1	F		D					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2.11

0	1	1	AC		0	1	0	F		D					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Fig. 2.12

conţinutului registrului tampon A, din dispozitivul cu adresa D, în acumulatorul AC din unitatea centrală. De menţionat că fiecare dispozitiv periferic are registre tampon (A, B, C) care participă la operaţii de intrare/ieşire. Din acest motiv există încă două instrucţiuni asemănătoare (DIB şi DIC) care se referă la registrele B şi respectiv C ale perifericului.

d) DOA Data Out A (Operaţie de ieşire în registrul tampon A). Formatul acestei instrucţiuni este dat în figura 2.12.

Executarea instrucţiunii determină transferarea conţinutului acumulatorului AC în registrul tampon al dispozitivului specificat prin cîmpul D şi executarea funcţiei specificată prin F. Cantitatea de date transferate în dispozitiv depinde de dimensinnea registrului tampon, de modul de operare al perifericului, etc. Conţinutul original al acumulatorului AC rămîne neschimbat.

2.3. PREZENTAREA GENERALĂ A SISTEMULUI DE OPERARE RDOS* NOVA

RDOS [63] este un sistem de operare în timp real, parţial rezident pe disc şi parţial rezident în memorie principală. Ca sistem de timp real cu acces multiplu el este astfel conceput încît poate să planifice şi să aloce controlul la mai multe sarcini de program diferite, sau la mai multe programe diferite, oferind posibilitatea utilizării simultane a resurselor sistemului.

Un sistem de timp real este un sistem astfel organizat încît permite prelucrarea informaţiei într-o manieră rapidă şi cu furnizarea rezultatelor într-un interval de timp prestabilit. Răspunsul dat de sistem trebuie să fie suficient de prompt astfel încît rezultatele prelucrării să fie disponibile în timp util pentru a putea influenţa procesul ce este supravegheat şi controlat.

Sistemul de operare RDOS este un sistem executiv proiectat pentru a realiza interfaţa cu programele de utilizator ce au necesităţi de timp real.

Subsistemul comunicaţiilor de intrare/ieşire împreună cu o planificare eficientă şi cu facilităţile de prelucrarea întreruperilor oferă o organizare satisfăcătoare pentru orice program de timp real.

Indiferent de forma de organizare în care apare, sistemul RDOS este un sistem de operare orientat *pe sarcini* de program (tasks), care admite ca unitate de planificare *lucrarea* alcătuită dintr-una sau mai multe sarcini.

Sarcina de program este o componentă logic independentă a programului, reprezentînd o etapă de execuţie asincronă în cadrul acestuia, etapă pentru care se alocă resursele fizice şi logice necesare.

O altă caracteristică a sistemului RDOS constă în faptul că este un sistem de operare condus de evenimente. Aceasta înseamnă că utilizatorul nu trebuie să folosească chemări speciale de sistem pentru a determina replanificarea

* Real Time Disk Operating System care în traducere înseamnă „Sistem de operare în timp real rezident pe disc".

sarcinilor. Sistemul menţine în execuţie sarcina cu prioritatea cea mai mare (prioritate definită de utilizator). Această sarcină va continua să fie în execuţie pînă la apariţia unuia din următoarele evenimente:

a) Sarcina îşi termină propriile-i operaţii.

b) Sarcina devine suspendată în timp ce aşteaptă apariţia unui eveniment (de exemplu, completarea unei operaţii de intrare/ieşire).

c) O sarcină de prioritate mai mare a devenit capabilă să treacă în execuţie.

Sistemul de operare RDOS permite utilizatorului să-şi implementeze aplicaţia pe un calculator rapid, folosind următoarele facilităţi:

a) Gestiunea standard a dispozitivelor de intrare/ieşire.

b) Program standard de deservirea întreruperilor.

c) Toate funcţiile necesare pentru a planifica utilizarea eficientă a unităţii centrale şi a perifericelor.

Sistemul RDOS poate apare într-un număr nelimitat de variante în funcţie de configuraţia „hardware" şi caracterul aplicaţiilor. Din acest motiv, la proiectarea sistemului s-a pus un accent deosebit pe modularitatea şi flexibilitatea sa.

Pentru a folosi toate facilităţile oferite de sistemul RDOS, utilizatorul are nevoie de o configuraţie minimă formată din: un calculator NOVA cu o memorie internă avînd o capacitate de cel puţin 16 000 de cuvinte, o consolă teletype sau un dispozitiv de afişaj cu tub catodic dotat cu claviatură şi un disc. În afara acestei configuraţii minime, RDOS poate gestiona memorie internă adiţională, discuri cu capete fixe cu capacitate de 4 milioane cuvinte, 8 pachete de discuri cu capete mobile, 16 deruloare de benzi magnetice (de 7 sau 9 piste), unităţi de casete magnetice, cititoare de cartele, imprimante, echipament de comunicaţie, ploter digital şi adaptoare pentru comunicaţii multiprocesor.

Sistemul de operare RDOS poate fi folosit atît interactiv, de la claviatura unei console, cît şi în mod secvenţial (batch mode) prin şiruri de lucrări lansate de la cititoare de cartele, sau de pe fişiere pe discuri, casete sau benzi magnetice.

Organizarea sistemului de operare RDOS

Componenta principală a sistemului RDOS este programul executiv, un program ce trebuie să fie rezident în memoria operativă înainte de a apare vreun proces de prelucrare. Funcţiile esenţiale realizate de acet program sînt: prelucrarea întreruperilor, administrarea bufferelor* şi a overlayului**, prelucrarea chemărilor de sistem şi deservirea întreruperilor de intrare/ieşire.

Alte module ale sistemului sînt aduse în memoria principală de pe disc, atunci cînd sînt chemate să execute funcţii specifice cum ar fi iniţializări parţiale sau complete ale sistemului sau ale dispozitivelor periferice, operaţii de întreţinerea fişierelor (deschidere, închidere, amendare, schimbarea denumirii), ş.a.

În figura 2.13 se află cel mai simplu model de organizare RDOS, cu un singur program de utilizator şi fără translatare în spaţiul de adrese virtuale.

---

* Prin buffer se înţelege o memorie tampon de capacitate mică şi cu acces rapid.

** Overlay este o tehnică folosită pentru aducerea unor rutine (părţi de program) de pe o memorie de masă în memoria principală, în timpul executării programului. Acest procedeu permite ca mai multe rutine să ocupe o aceeaşi zonă din memoria principală, la momente de timp diferite şi se foloseşte pentru a realiza economii în utilizarea spaţiului de memorie operativă.

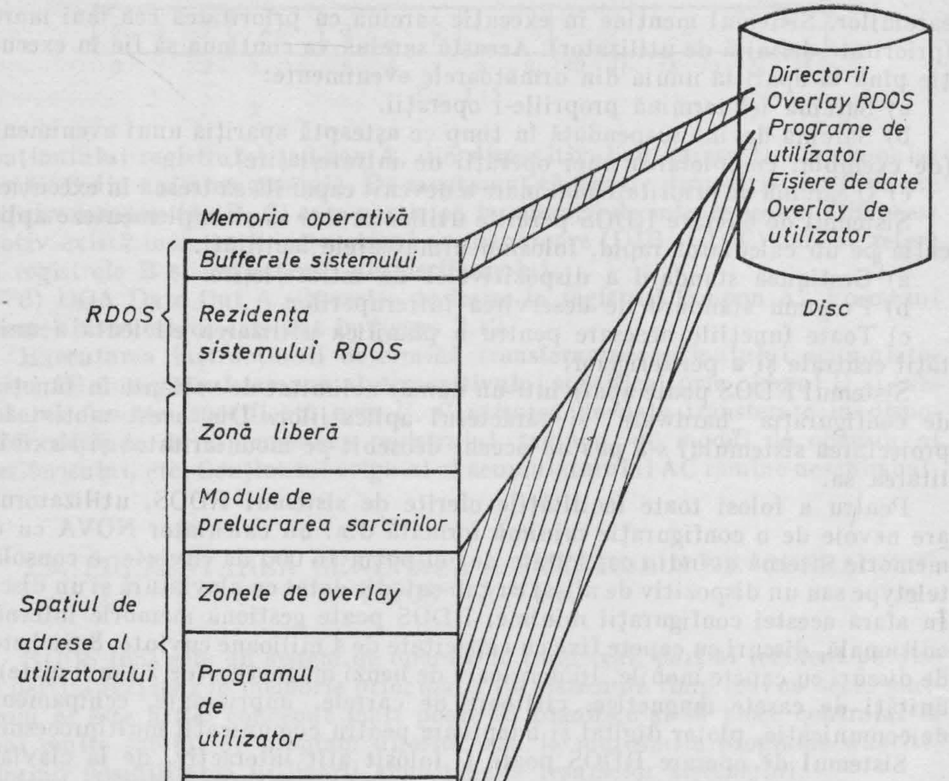


Fig. 2.13

În această variantă de organizare pagina zero a utilizatorului include adresa 16 și apoi se extinde de la locația 20 la 377. La adresa 16 se află o locație specială rezervată de către sistem pentru a fi folosită ori de cîte ori controlul programului este transferat de la o sarcină la alta. Locația 17 este de asemenea rezervată de către sistemul de operare pentru a fi folosită în prelucrarea chemărilor de sistem.

Tabela de stări a utilizatorului începe la prima adresă de memorie normal relocatabilă, 400_8 și merge pînă la 421_8. Ea conține informații ce descriu programul de utilizator, cum ar fi: lungimea programului, numărul de sarcini și de canale de intrare/ieșire active specificate de către program, precum și alte informații permanente sau variabile. În continuarea tabelei de stări se înscrie programul de utilizator propriu zis, în cadrul căruia pot fi intercalate una sau mai multe zone de „overlay" ale utilizatorului. Aceste zone primesc por-

ţiuni rezidente pe disc ale programului de utilizator, atunci cînd acestea sînt apelate prin chemări de sarcini sau de sistem. În continuarea programului se află blocul de control a sarcinilor (TCB — Task Control Block) şi module din biblioteca sistemului (SYB.LB. — System Library) TCB este apelat de către sistem pentru a înregistra starea curentă a informaţiei variabile legate de fiecare sarcină activă din cadrul programului. Modulele bibliotecii de sistem sînt necesare pentru a asista chemările multisarcini (multitask) editate de către program. De asemenea, din biblioteca de sistem se încarcă planificatorul multisarcină care controlează operarea sarcinilor.

În cadrul unui sistem de operare în timp real, necesităţile individuale ale utilizatorului pot varia fie din cauza configuraţiei hardware, fie din cauza caracterului diferit al aplicaţiilor şi al facilităţilor necesitate. Ca urmare fiecare sistem de operare va trebui să fie proiectat în funcţie de necesităţile specifice şi de configuraţia hardware existentă. Această funcţie de aplicare a unui sistem de operare la nişte condiţii concret existente, se numeşte generarea sistemului de operare. Procesul de generarea sistemului oferă posibilitatea creerii unui monitor compus din programe şi subrutine de firmă şi de utilizator. Produsul final al generării sistemului este un sistem de operare rezident pe disc care a fost adaptat la necesităţile utilizatorului, devenind eficient pentru o anumită configuraţie de echipament şi pentru funcţii de operare specifice.

Segmentarea programului de utilizator

Dacă spaţiul de adrese al utilizatorului nu este suficient de mare pentru a conţine programul în întregime sistemul de operare permite segmentarea programului în mai multe părţi de dimensiuni fixe, plasarea unora dintre ele pe disc şi aducerea lor în memoria principală (în zone prestabilite), la momentul execuţiei.

Aceste segmente de program ce sînt vehiculate de pe disc se numesc componente de „overlay" ale utilizatorului şi se află memorate pe disc sub formă de imagine-memorie pentru a oferi posibilitatea unei încărcări rapide la momentul execuţiei. Însăşi sistemul de operare foloseşte procedeul de „overlay" pentru propriile-i componente; acestea se numesc componente „overlay" de sistem pentru a putea fi deosebite de cele ale utilizatorului.

Un set complet de componente „overlay" ale utilizatorului folosite de către un anumit program poartă denumirea de fişier „overlay". Acest fişier este organizat contiguu. Organizarea contiguă, împreună cu formatul de imagine memorie fac ca încărcarea unei componente „overlay" la un program să se realizeze rapid şi eficient.

Executarea duală a programelor

Pentru a mări gradul de utilizare al sistemului, RDOS permite rezidenţa a două programe în memoria operativă şi derularea lor într-o simultaneitate aparentă. Aceasta înseamnă că sistemul de operare comută controlul între două planificatoare de sarcini în conformitate cu o prioritate predeterminată, stabilită de către utilizator. Pentru a se proteja integritatea ambelor programe,

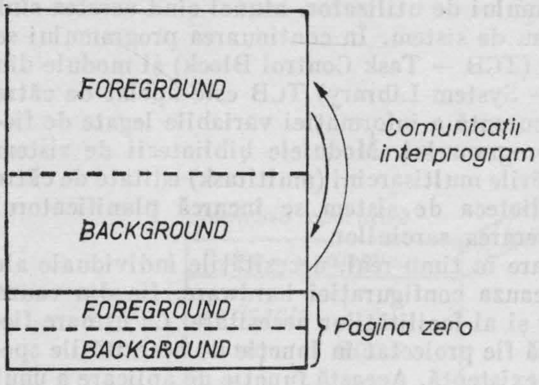


Fig. 2.14

acestea sînt plasate în două partiții de memorie distincte denumite „background și „foreground", așa după cum se vede în figura 2.14.

Pagina zero a memoriei este utilizată în comun de cele două partiții.

Programarea în „foreground — background" permite executarea aparent simultană a unor colecții de sarcini distincte folosind împărțirea resurselor sistemului.

Un exemplu tipic de aplicație „foreground-background" sub sistemul de operare RDOS constă în prelucrarea unui program de control a unui proces în timp real, în partiția „foreground" și compilarea și asamblarea altui program în partiția „background". Deși programele din cele două partiții sînt complet independente ele pot comunica editînd mesaje prin intermediul unor chemări de sistem sau prin adaptorul de comunicații multiprocesor.

Programarea în sistemul „foreground/background" cu partiționarea software a memoriei este în mod cert o utilizare mai eficientă a resurselor calculatorului decît în cazul executării unui singur program. Cu toate acestea, chiar și această schemă îmbunătățită de utilizare prezintă unele dezavantaje. Primul dezavantaj constă în aceea că nu există nici o garanție că un program adresînd din greșeală o locație din afara spațiului de adrese ce i-a fost alocat, nu va aduce prejudicii programului înscris în spațiul de memorie violat. Pe de altă parte fiecare program trebuie astfel dimensionat încît să poată încape în partiția respectivă. În plus pagina zero trebuie împărțită în timp între cele două partiții, fapt ce micșorează resursele disponibile fiecărei partiții. Aceste dezavantaje pot fi depășite prin implementarea unei opțiuni hardware denumită „Unitate de protecție și administrare a memoriei" (Memory Management and Protection Unit), aplicabilă la calculatoarele din familia NOVA 800 și SUPERNOVA, [61].

La aceste calculatoare separarea între partițiile de memorie „background, foreground" și sistemul de operare se face prin hardware. În afară de aceasta, opțiunea de administrare a memoriei extinde spațiul maxim de memorie internă, pentru o singură unitate centrală, de la 32 K* cuvinte, la spații pînă la 31 de K cuvinte pentru fiecare din partițiile alocate foregroundului, backgroundului sau rezidenței sistemului de operare. Acest lucru se realizează printr-un sistem de translatare de la spațiul logic la spațiul fizic de adrese, sistem prezentat în figura 2.15. Această translatare se realizează la nivel de pagină formată din 1024 de adrese (1 K cuvînt). Se observă că într-o astfel de organizare atît foregroundul cît și backgroundul dispun de cîte o pagină zero proprie.

---

* 1K = 1024 cuvinte.

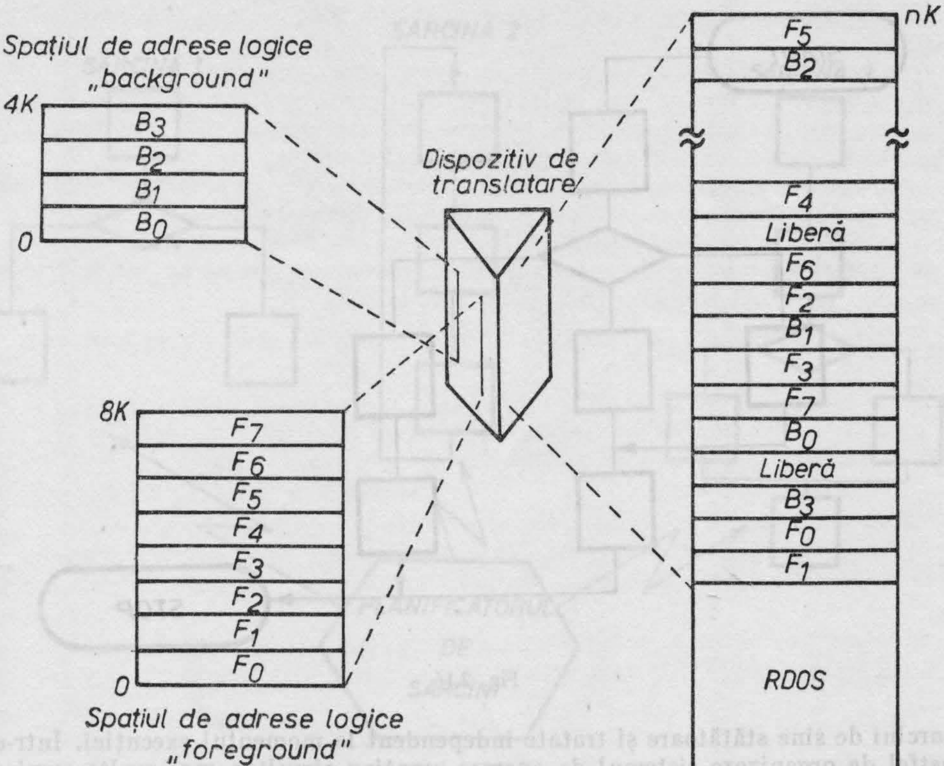


Fig. 2.15

Organizarea disciplinei multisarcină (multitask). Stările şi priorităţile sarcinilor

Dacă sistemul de operare nu poate recunoaşte la un moment dat decît o singură sarcină activă, modul de lucru se numeşte cu organizare monosarcină (single task). O astfel de organizare este prezentată în figura 2.16 şi este caracterizată prin faptul că logica programului este parcursă într-un singur pas, indiferent de complexitatea ei. Organizarea monosarcină este specifică sistemelor ce nu deservesc aplicaţii de timp real. Programele scrise în FORTRAN IV, programe de utilizator scrise în assembler, compilatoarele, asambloarele, încărcătoarele relocatabile, editoarele, sînt toate exemple de programe ce se execută în disciplina monosarcină. În organizarea monosarcină unitatea centrală devine inactivă pe timpul cît programul iniţiază o operaţie de intrare-ieşire. Programul este reactivat în momentul în care operaţia a fost terminată. Se pierde astfel un timp însemnat din capacitatea de lucru a unităţii centrale. Pe de altă parte trebuie ţinut seama că multe din aplicaţiile utilizatorului conţin părţi independente ca structură logică. Aceste părţi pot fi considerate ca

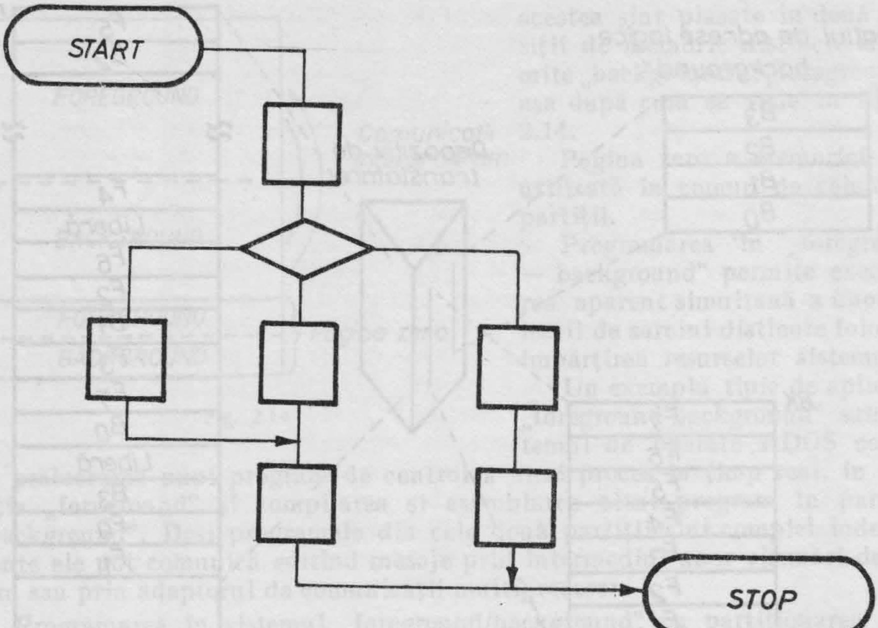


Fig. 2.16

sarcini de sine stătătoare şi tratate independent la momentul execuţiei. Într-o astfel de organizare sistemul de operare menţine simultan mai multe sarcini active, controlul unităţii centrale fiind alocat de către *planificatorul de sarcini*, la sarcina cu cea mai mare prioritate care este pregătită pentru execuţie. Fiecare sarcină efectuează asincron o funcţie specifică în timp real.

Acest mod de lucru poartă denumirea de organizare multisarcină (multitask) şi este prezentat în figura 2.17. Exemple de organizare multisarcină: aplicaţiile de rezervarea locurilor în transportul aerian, operaţii de control a unui proces, prelucrarea mesajelor venite de la diverse terminale cînd se lucrează în teleprelucrare, deservirea mesajelor de interogare a unor fişiere sau bănci de date, etc. O eficienţă sporită se obţine, prin aplicarea modului de lucru multisarcină şi în cazul rutinelor de interpretare şi editare a textului simbolic în limbajul BASIC. Primele versiuni ale acestor programe permiteau deservirea unui singur utilizator pe durata lucrării sale. În prezent prin aplicarea organizării multisarcină, sistemul de operare menţine simultan mai multe sarcini active, ceea ce conduce la scăderea considerabilă a timpului de răspuns aferent mesajelor introduse de la consolele utilizatorilor, [67].

Lansarea în execuţie a unui program de utilizator, în disciplina multisarcină, sub control RDOS, se face pornind de la o sarcină specială, desemnată de către utilizator şi denumită „default task", care are proprietatea că se autoiniţiază în lipsa altei sarcini active în sistem. În continuare, din cadrul acestei sarcini, utilizatorul poate iniţia alte sarcini lansînd chemări corespunzătoare. Dacă la un moment dat se află mai multe sarcini active în sistem, este necesară intervenţia planificatorului de sarcini care decide ce sarcină va fi lansată în execuţie.

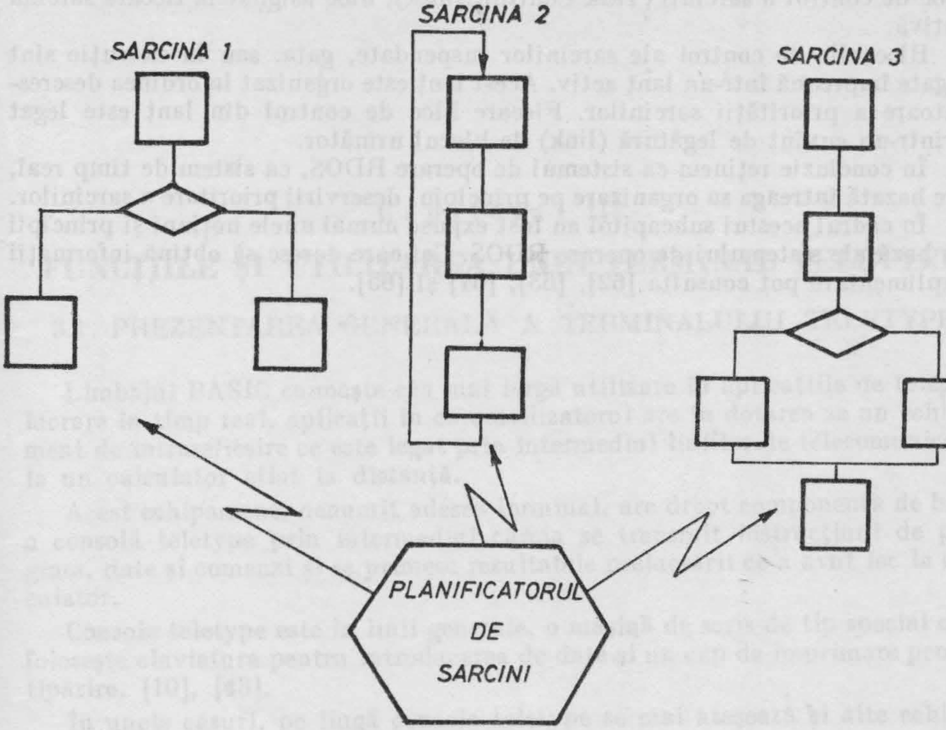


Fig. 2.17

Planificatorul ia decizie în funcţie de priorităţile pe care le posedă sarcinile la momentul iniţierii. Pot exista 256 de nivele de priorităţi diferite, în gama 0 ... 255, prioritatea zero fiind de nivel maxim. Planificatorul predă întotdeauna controlul unităţii centrale către sarcina cu prioritatea cea mai mare şi care este pregătită pentru execuţie. O sarcină se poate afla în sistem în una din următoarele patru stări:

a) EXECUŢIE. Sarcina se află sub controlul unităţii centrale.

b) GATA. Sarcina este pregătită pentru execuţie dar nu poate primi controlul unităţii centrale pînă cînd toate sarcinile de prioritate mai mare aflate în stările GATA sau EXECUŢIE sînt terminate sau trec în starea de SUSPENDARE.

c) SUSPENDATĂ. Sarcina aşteaptă apariţia sau completarea unei chemări de sistem sau a altei operaţii de timp real.

d) INACTIVĂ. Sarcina nu a fost iniţiată, deci nu este cunoscută de către sistemul de operare sau execuţia ei a fost terminată şi prin aceasta a devenit inactivă.

Primele trei stări sînt considerate stări în care sarcina este activă deoarece ea se află în evidenţa şi sub controlul sistemului de operare. Cînd o sarcină este activă, ea trebuie să fie definită în orice moment prin informaţii de stare pentru a permite planificatorului să menţină în stare de execuţie sarcina cu prioritatea cea mai mare. Această informaţie de stare se află înscrisă într-un

bloc de control a sarcinii (Task Control Block), bloc asignat la fiecare sarcină activă.

Blocurile de control ale sarcinilor suspendate, gata, sau în execuţie sînt legate împreună într-un lanţ activ. Acest lanţ este organizat în ordinea descrescătoare a priorităţii sarcinilor. Fiecare bloc de control din lanţ este legat printr-un cuvînt de legătură (link) de blocul următor.

În concluzie reţinem că sistemul de operare RDOS, ca sistem de timp real, are bazată întreaga sa organizare pe principiul deservirii prioritare a sarcinilor.

În cadrul acestui subcapitol au fost expuse numai unele noţiuni şi principii de bază ale sistemului de operare RDOS. Cei care doresc să obţină informaţii suplimentare pot consulta [62], [63], [64] şi [65].

Capitolul III

FUNCȚIILE ȘI UTILIZAREA UNUI TERMINAL TELETYPE

3.1. PREZENTAREA GENERALĂ A TERMINALULUI TELETYPE

Limbajul BASIC cunoaşte cea mai largă utilizare în aplicaţiile de teleprelucrare în timp real, aplicaţii în care utilizatorul are în dotarea sa un echipament de intrare/ieşire ce este legat prin intermediul liniilor de telecomunicaţii la un calculator aflat la distanţă.

Acest echipament, denumit adesea terminal, are drept componentă de bază o consolă teletype prin intermediul căreia se transmit instrucţiuni de program, date şi comenzi şi se primesc rezultatele prelucrării ce a avut loc la calculator.

Consola teletype este în linii generale, o maşină de scris de tip special care foloseşte claviatura pentru introducerea de date şi un cap de imprimare pentru tipărire, [10], [43].

În unele cazuri, pe lîngă consola teletype se mai ataşează şi alte echipamente de intrare şi ieşire (în general un cititor de cartele şi o imprimantă rapidă), iar terminalul poartă denumirea de „terminal greu".

Terminalul teletype cunoaşte mai multe variante, dintre care cele mai răspîndite sînt KSR (Keyboard Send-Receive) şi AST (Automatic Send-Receive). În figurile 3.1 şi 3.2 se prezintă schemele bloc, cuprinzînd componentele de bază ale celor două tipuri de terminale.

Se observă că terminalul ASR are în plus faţă de terminalul KSR, un cititor şi un perforator de bandă.

Dispozitivul de imprimare are ca element principal un cap de imprimare de formă cilindrică sau tronconică pe care sînt plasate circular, pe mai multe rînduri, toate caracterele claviaturii.

Capul de imprimare poate executa o mişcare de rotaţie de 180° la stînga şi la dreapta precum şi o deplasare în plan vertical într-un număr de trepte egal cu numărul de rînduri de caractere.

Datorită acestui fapt se poate selecta la un moment dat, în poziţia de imprimare, orice caracter din repertoriu.

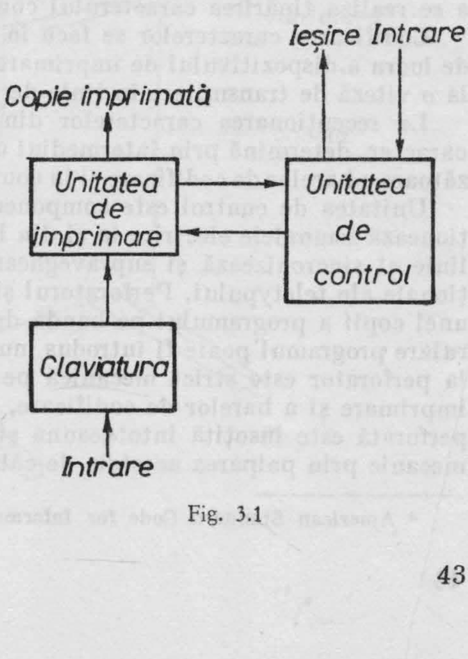


Fig. 3.1

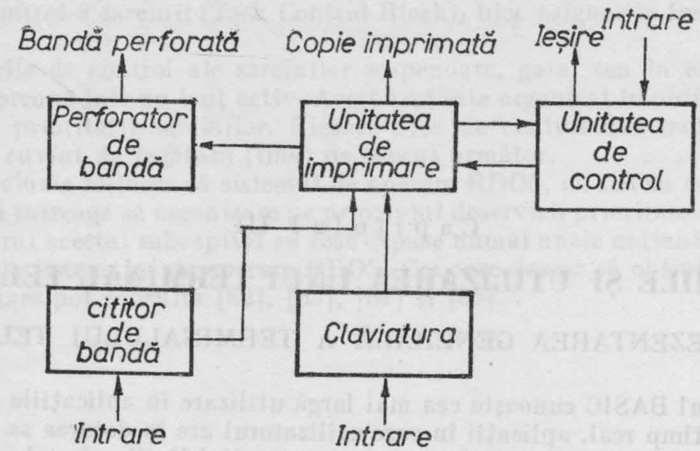


Fig. 3.2

În funcţionarea lor, claviatura şi dispozitivul de imprimare utilizează în comun, ca element de bază, un sistem de codificare mecanică.

Acest sistem este format din mai multe bare longitudinale, paralele, peste care sînt plasate nişte bare transversale, de asemenea paralele.

Pe barele longitudinale sînt decupate nişte profile conform unui anumit sistem de codificare. Barele transversale sînt legate mecanic prin intermediul unor leviere la tastele claviaturii.

În momentul acţionării unei taste, levierul transmite mişcarea mecanică la bara transversală, iar aceasta selectează anumite bare longitudinale, fapt ce determină închiderea unor contacte electrice reprezentînd codul caracterului respectiv. Această codificare paralelă este convertită apoi în formă serială prin intermediul unui distribuitor electro-mecanic, formă ce se transmite în linie. De asemenea, codul electric corespunzător este transmis şi dispozitivului de imprimare care selectează poziţia necesară a capului de imprimare pentru a se realiza tipărirea caracterului corespunzător.

Codificarea caracterelor se face în cod ASCII* cu 8 biţi /caracter. Viteza de lucru a dispozitivului de imprimare este de 10 char/sec., ceea ce corespunde la o viteză de transmitere în linie de 110 bauds.

La recepţionarea caracterelor din linie, semnalele corespunzătoare unui caracter, determină prin intermediul unor electromagneţi o selectare corespunzătoare a barelor de codificare şi în continuare, a poziţiei capului de imprimare.

Unitatea de control este componenta teletypului care transmite şi recepţionează semnalele electrice în şi din linie. Ea decodifică comenzile venite din linie şi sincronizează şi supraveghează funcţionarea celorlalte blocuri funcţionale ale teletypului. Perforatorul şi cititorul de bandă permit înregistrarea unei copii a programului pe bandă de hîrtie perforată, astfel încît la o nouă rulare programul poate fi introdus mult mai rapid, direct din bandă. Intrarea la perforator este strict mecanică pe baza unei extensii a dispozitivului de imprimare şi a barelor de codificare, astfel că realizarea unei copii pe bandă perforată este însoţită întotdeauna şi de imprimare. Citirea benzilor se face mecanic prin palparea acesteia de către un set de perii.

* American Standard Code for Information Interchange.

Combinaţia de cod corespunzătoare unui caracter este reprezentată pe bandă transversal, prin 8 canale în care se află sau nu perforaţii. Între cele 8 canale ce materializează informaţia codificată binar, există un canal format dintr-un şir continuu de perforaţii, cu diametrul mai mic, utilizat pentru antrenarea benzii şi pentru sincronizare.

Fiecare perie urmăreşte un canal, deci un anumit rang binar din combinaţia de cod.

În momentul în care se întîlneşte o perforaţie, peria face contact cu suprafaţa metalică pe care se mişcă banda şi închide un circuit electric, generînd un impuls. Se materializează astfel cifra „1 binar" dintr-o combinaţie de cod asignată unui caracter. În lipsa perforaţiei peria nu închide circuitul, nu se generează impuls şi pe poziţia periei corespunzătoare rangului binar din codul caracterului va apare lipsă de impuls, deci cifra „0 binar".

Ieşirea din dispozitivul de citire a benzii este legată în paralel la unitatea de control pentru transmiterea informaţiei în linie, cît şi la distribuitorul unităţii de tipărire pentru a se realiza imprimarea în clar, la consolă, a informaţiei citite de pe bandă.

În fig. 3.3 se prezintă imaginea unei console teletype ASR. Claviatura are tastele aşezate pe patru rînduri şi în plus o tastă specială (mai lungă) pentru

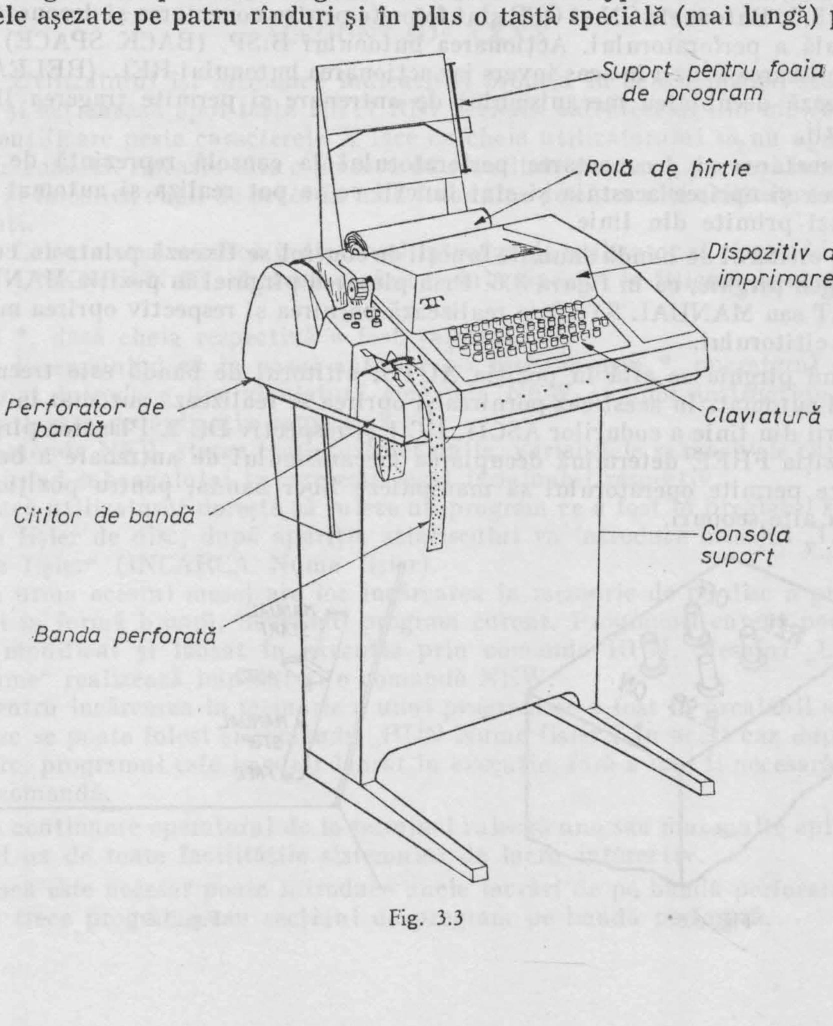


Fig. 3.3

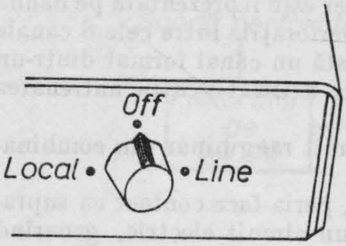


Fig. 3.4

spaţiere. Unele dintre taste permit introducerea a două caractere diferite. Pe tastă cele două caractere sînt scrise unul sub altul.

În mod normal, la acţionarea tastei este introdus caracterul de jos. Pentru introducerea caracterului de deasupra, concomitent cu acţionarea tastei respective se va acţiona şi tasta SHIFT.

În partea din dreapta a panoului frontal se află comutatorul de punere sub tensiunea care are trei poziţii, ca în figura 3.4. În poziţia „OFF" consola teletype este scoasă de sub tensiune. În poziţia „LOCAL" consola este pusă sub tensiune dar nu poate realiza decît funcţii locale (off line) iar în poziţia „LINE" este pusă sub tensiune şi legată la linie.

După cum se observă din fig. 3.3, în partea din stînga a consolei se află plasate perforatorul şi cititorul de bandă. Fiecare dintre acestea pot fi trecute în diferite regimuri de lucru, prin comenzi date de la butoane sau comutatoare. Astfel, perforatorul de bandă este prevăzut cu un grup de 4 butoane ca în figura 3.5. Butoanele ON şi OFF sînt folosite pentru conectarea şi deconectarea manuală a perforatorului. Acţionarea butonului B.SP. (BACK SPACE) permite mişcarea benzii în sens invers iar acţionarea butonului REL. (RELEASE) realizează decuplarea mecanismului de antrenare şi permite tragerea liberă a benzii.

Conectarea şi deconectarea perforatorului la consolă reprezintă de fapt pornirea şi oprirea acestuia şi sînt funcţii ce se pot realiza şi automat prin comenzi primite din linie.

La cititorul de bandă anumite funcţii de control se fixează printr-un comutator gen pîrghie, ca în figura 3.6. Prin plasarea pîrghiei în poziţia MANUAL START sau MANUAL STOP se realizează pornirea şi respectiv oprirea manuală a cititorului.

Cînd pîrghia se află în poziţia AUTO, cititorul de bandă este trecut în modul automat. În acest caz pornirea şi oprirea se realizează automat în urma primirii din linie a codurilor ASCII, DC 1 şi respectiv DC 2. Plasarea pîrghiei în poziţia FREE determină decuplarea mecanismului de antrenare a benzii, fapt ce permite operatorului să manipuleze liber banda, pentru poziţionare sau în alte scopuri.

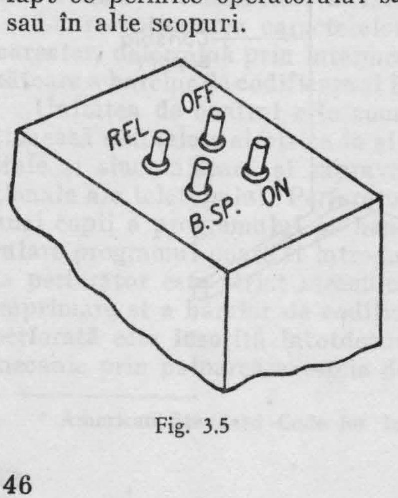


Fig. 3.5

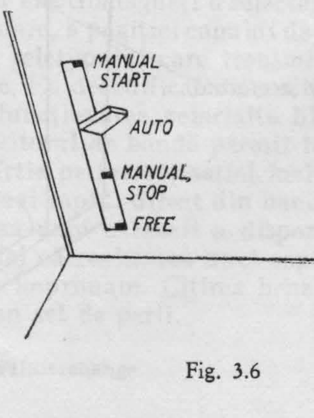


Fig. 3.6

Cititorul este prevăzut şi cu nişte martori electromecanici care comandă oprirea automată în cazul lipsei de bandă sau cînd apar defecţiuni în antrenarea fluentă a acesteia.

Unele console teletype sînt dotate cu facilităţi de apel şi răspuns automat, facilităţi ce sînt utilizate pentru stabilirea legăturilor pe linii comutate (vezi cap. I). În acest caz pe panoul frontal al consolei se află un disc telefonic şi mai multe butoane şi lămpi de semnalizare destinate acestui scop.

Accesul utilizatorului de la terminal la resursele fizice şi logice ale calculatorului are loc conform unui protocol de identificare a acestuia.

Timpul în care un terminal foloseşte resursele sistemului de calcul aflat la distanţă se numeşte *sesiune de terminal* şi poate fi considerat analogul lotului de lucrări din regimul secvenţial (batch processing). Vom prezenta în continuare un model de desfăşurare a unei sesiuni de terminal. Presupunînd că legătura fizică între terminal şi calculator a fost deja stabilită, se parcurg apoi următoarele etape:

1. Se trece comutatorul din fig. 3.4 în poziţia „LINE".
2. Se deschide sesiunea de terminal acţionînd tasta ESC de pe claviatură.
3. Calculatorul verifică identitatea utilizatorului tipărind mesajul:

ACCOUNT-ID: XXXX

4. Utilizatorul îşi introduce indicativul propriu în cele 4 poziţii marcate cu X şi acţionează apoi tasta RETURN. Scrierea caracterelor din indicativul de identificare peste caracterele X face ca cheia utilizatorului să nu apară în clar la consolă. Aceasta este o măsură de securitate pentru a preveni interceptarea şi folosirea cheii de acces în mod fraudulos de către alţi utilizatori neautorizaţi.
5. Calculatorul verifică indicativul introdus de utilizator şi răspunde prin:

a) UNKNOWN ID, dacă nu a găsit codul respectiv în fişierul de chei, sau prin

b) *, dacă cheia respectivă a fost validată.

6. Presupunînd că în pasul anterior s-a răspuns prin *, operatorul de la terminal introduce din claviatură comanda NEW şi apoi instrucţiunile şi datele legate de aplicaţia respectivă.

Comanda NEW şterge toate instrucţiunile, variabilele şi masivele curente, aparţinînd subcanalului la care este legat terminalul respectiv.

Dacă utilizatorul doreşte să ruleze un program ce a fost în prealabil salvat pe un fişier de disc, după apariţia asteriscului va introduce mesajul „LOAD Nume fişier" (INCARCA Nume fişier).

În urma acestui mesaj are loc încărcarea în memorie de pe disc a programului în formă binară, devenind program curent. Programul curent poate fi apoi modificat şi lansat în execuţie prin comanda RUN. Mesajul „LOAD filename" realizează implicit şi o comandă NEW.

Pentru încărcarea în memorie a unui program ce a fost în prealabil salvat pe disc se poate folosi şi comanda „RUN Nume fişier". În acest caz după încărcare, programul este imediat lansat în execuţie, fără a mai fi necesară vreo altă comandă.

În continuare operatorul de la terminal rulează una sau mai multe aplicaţii făcînd uz de toate facilităţile sistemului de lucru interactiv.

Dacă este necesar poate introduce unele lucrări de pe bandă perforată sau poate trece programe sau secţiuni de program pe bandă perforată.

De asemenea, utilizînd comanda „SAVE Nume fişier", programul curent poate fi scris în format binar pe un disc magnetic.

7. Închiderea sesiunii de terminal este marcată de tastarea comenzii BYE, la care calculatorul răspunde prin mesajul **READY**.

Mesajele şi comenzile din modelul de sesiune expus mai sus nu sînt standardizate. Ele depind de sistemul de operare utilizat, dar etapele sesiunii sînt în general aceleaşi.

În continuare vom prezenta utilizarea unor taste şi comenzi de consolă, în cadrul limbajului BASIC.

3.2. UTILIZAREA CLAVIATURII

Claviatura consolei este folosită pentru introducerea instrucţiunilor de program cît şi pentru realizarea altor comenzi legate de încărcarea, executarea şi listarea programului.

Introducerea programului la consolă se face instrucţiune cu instrucţiune, conform regulilor de sintaxă specifice limbajului. Pe o linie de consolă se scrie o singură instrucţiune, iar la terminarea tipăririi acesteia se acţionează tasta RETURN. Pentru a uşura redactarea, acţionarea tastei RETURN va fi descrisă în continuare prin simbolul ↵).

Întrucît blancul nu este un separator în cadrul limbajului, numărul de spaţii libere între cuvintele unei instrucţiuni este opţional. În continuare se vor analiza funcţiile realizate de acţionarea unor taste sau de tipărirea unor cuvinte de comandă, [65].

Funcţiile tastelor de control

Tasta ESC (Escape)

Acţiunea tastei ESC înseamnă în general „întreruperea operaţiei curente". Efectul depinde de la caz la caz de starea sistemului, după cum urmează:

a) Dacă programul se afla în execuţie şi nu întîlnise nici o instrucţiune de forma ON ESC THEN ... , execuţia încetează şi sistemul afişează mesajul:

STOP AT ****,

unde **** reprezintă eticheta ultimei instrucţiuni executate. Sistemul trece în modul „Keyboard", adică aşteaptă să primească comenzi sau instrucţiuni de la consolă.

b) Dacă programul se afla în execuţie şi întîlnise mai înainte o instrucţiune de forma ON ESC THEN, controlul programului va fi transferat la această instrucţiune.

c) Dacă se executa o comandă dată de la claviatură, aceasta se termină şi apoi sistemul aşteaptă alte mesaje de la claviatură.

d) Dacă sistemul era inactiv (idle mode), acţionarea tastei ESC activează terminalul pentru operare. Sistemul devine inactiv după ce utilizatorul editează o comandă BYE (vezi 3.1). În concluzie, ori de cîte ori utilizatorul doreşte să editeze o comandă şi sistemul se află într-unul din modurile enumerate mai sus, acţionarea tastei ESC îl va trece în modul „Keyboard", mod în care acceptă mesaje din claviatură.

Tasta **L SHIFT**

Acţionarea tastei L în conjuncţie cu tasta SHIFT determină ştergerea din memorie a ultimei linii introduse din consolă. Manevra este folosită cînd în scrierea unei instrucţiuni sau a unor date cerute de către o instrucţiune INPUT s-au comis greşeli şi utilizatorul doreşte să anuleze linia. Pentru a confirma ştergerea, sistemul tipăreşte la sfîrşitul liniei simbolul / şi poziţionează carul la începutul unei linii noi. *Exemplu:*

```
20 PRONT "VALPA/"
20 PRINT "VALOAREA LUI X ESTE:" ; X
```

Tasta **RUBOUT**

Acţionarea acestei taste este folosită tot pentru efectuarea unor ştergeri în memorie, dar de data aceasta la nivel de caracter.

Sistemul confirmă ştergerea prin tipărirea simbolului ←, după care utilizatorul va introduce caracterele corecte.

Exemplu:

```
90 PRO←INT "VALPA← ←OAREA LUI X ESTE:" ; X
```

Comenzi de modificarea programului

Utilizatorul poate modifica structura programului său ştergînd, inserînd sau schimbînd una sau mai multe instrucţiuni. Comenzile de claviatură ce realizează aceste funcţii se bazează pe etichetele instrucţiunilor şi arată astfel:

Forma comenzii	*Acţiunea*
n↵	Sistemul caută în programul curent instrucţiunea cu etichete „n" şi o şterge. Dacă în program nu există o astfel de etichetă nu se realizează nici o acţiune.
n_1, n_2↵	Sistemul şterge din programul curent un cîmp de instrucţiuni începînd cu cea care are eticheta „n_1" şi terminînd cu cea care are eticheta „n_2".
n_1, ↵	Sistemul şterge din programul curent un cîmp de instrucţiuni începînd cu cea care are eticheta „n_1" şi terminînd cu ultima instrucţiune de program.
, n_2 ↵	Sistemul şterge din programul curent un cîmp de instrucţiuni începînd cu prima instrucţiune de program şi terminînd cu cea care are eticheta „n_2".
n Instrucţiune ↵	Sistemul inserează instrucţiunea în program la eticheta „n". Dacă în prealabil exista o altă instrucţiune cu această etichetă, ea va fi ştearsă şi noua instrucţiune îi va lua locul.

Comenzi de execuţie

Comanda **RUN**

Determină lansarea în execuţie a programului sau a unor părţi de program. De asemenea ea poate realiza şi încărcarea programului de pe disc sau de pe

un alt dispozitiv, urmată de lansarea lui în execuţie. Poate fi folosită în unul din următoarele formate:

a) RUN ↵
b) RUN etichetă ↵
c) RUN nume de fişier ↵

Efectele pentru fiecare caz în parte sînt:

a) Şterge toate locaţiile de variabile, reiniţializează pointerul din zona de date (vezi cap. 6), iniţializează generatorul de numere aleatoare şi lansează în execuţie programul curent începînd cu prima instrucţiune.

b) Toată informaţia existentă (valori de variabile, dimensionări de masive etc.) rezultată din execuţia anterioară a programului curent este păstrată iar programul este lansat în execuţie începînd de la eticheta care urmează comenzii RUN.

Această formă a comenzii RUN permite utilizatorului să reia execuţia programului de la un anumit punct reţinînd valorile curente ale tuturor variabilelor şi parametrilor obţinute în timpul execuţiei anterioare. Comanda se foloseşte mai ales după o instrucţiune STOP sau după un mesaj de eroare şi poate fi precedată de orice modificări aduse în program prin alte comenzi de consolă. Ea este o formă de manifestare a modului de lucru interactiv.

c) Şterge programul curent din partiţia utilizatorului şi încarcă un nou program de pe dispozitivul indicat prin „nume de fişier". De asemenea determină şi lansarea în execuţie a noului program începînd cu prima instrucţiune.

Comanda CON

Are formatul: CON ↵. Determină restartarea programului dintr-un punct în care acesta şi-a întrerupt execuţia deoarece a întîlnit o instrucţiune STOP sau pentru că a apărut o eroare de semantică la o anumită instrucţiune. Acţiunea acestei comenzi este similară cu cea a comenzii „RUN etichetă", numai că în cazul lui CON, nu este necesară specificarea etichetei, ea rezultînd implicit. Punctul de la care se reia execuţia programului este prima instrucţiune de după STOP sau instrucţiunea la care s-au semnalat erori, după ce acestea au fost corectate.

Comenzi de listare

Comanda LIST

Are ca efect listarea în cod ASCII a programului curent sau a unor părţi de program, la consolă sau pe un dispozitiv indicat prin „nume de fişier". Comanda se poate da în mai multe formate, cu efecte diferite, după cum urmează:

a) LIST
b) LIST etichetă
c) LIST TO etichetă
d) LIST etichetă e_1 {TO / ,} eticheta e_2

} [„nume de fişier"] ↵

a) Listează întregul program începînd cu prima instrucţiune.

b) Listează numai instrucţiunea a cărei etichetă a fost specificată.

c) Listează o parte de program începînd cu prima instrucţiune şi terminînd cu cea specificată prin etichetă.

d) Listează o parte de program, începînd cu instrucţiunea de la eticheta e_1 şi terminînd cu instrucţiunea de la eticheta e_2.

Dacă nu se specifică niciun nume de fişier listarea se va face la consolă. Exemple de comenzi LIST:

1. LIST 700 TO 900 ↵

Se vor lista la consolă instrucţiunile avînd etichetele cuprinse în gama 700—900 (inclusiv limitele).

2. LIST 80 ↵

Se va lista la consolă instrucţiunea avînd eticheta 80.

3. LIST "$LPT" ↵

Se va lista întregul program la imprimantă deoarece $LPT reprezintă numele de fişier al imprimantei.

4. LIST TO 400 "F 2.2" ↵

Se vor lista instrucţiunile de program începînd cu prima şi terminînd cu cea care are etichetă 400, pe fişierul F 2.2.

Comanda PUNCH

Este echivalentă cu comanda LIST, deosebindu-se de aceasta prin faptul că listarea se face întotdeauna la perforatorul de bandă al terminalului. Formatul comenzii este următorul:

a) PUNCH ↵

b) PUNCH etichetă ↵

c) PUNCH TO etichetă ↵

d) PUNCH eticheta e_1 $\left\{\begin{matrix}\text{TO}\\ ,\end{matrix}\right\}$ eticheta e_2 ↵

Semnificaţia fiecărei comenzi se deduce imediat prin analogie cu formatele comenzii LIST.

Listarea pe banda de hîrtie începe şi se încheie cu un număr de caractere nule.

De menţionat că editarea comenzii PUNCH nu determină pornirea perforatorului la terminal. Din acest motiv este necesar ca operatorul să parcurgă următoarea procedură:

1. Se tipăreşte la consolă comanda PUNCH, urmată de reîntoarcerea carului şi apoi se apasă imediat butonul ON de la perforatorul de bandă.

2. Sistemul va perfora un număr de caractere nule, listarea programului şi apoi un nou număr de caractere nule.

3. Cînd perforarea s-a terminat se acţionează butonul OFF de la perforatorul de bandă.

Comenzi de reetichetare a programului

Comanda RENUMBER

Permite reetichetarea tuturor instrucţiunilor din programul curent şi are următoarele formate:

a) RENUMBER ↵
b) RENUMBER n ↵
c) RENUMBER STEP n ↵
d) RENUMBER n_1 STEP n_2 ↵

În cadrul comenzilor de mai sus n, n_1 şi n_2 sînt numere naturale. Semnificaţia comenzilor este următoarea:

a) Are loc o reetichetare a programului, prima instrucţiune primind eticheta 10 iar următoarele etichete rezultate din incrementarea precedentei cu 10.

b) Are loc o reetichetare a programului, prima instrucţiune primind eticheta „n" iar următoarele etichete rezultînd din incrementarea cu 10.

c) Are loc o reetichetare a programului, prima instrucţiune primind eticheta 10 iar următoarele etichete rezultînd din incrementarea cu „n".

d) Are loc o reetichetare a programului, prima instrucţiune primind eticheta „n_1", iar celelalte etichete rezultînd din incrementarea cu „n_2".

Dacă în urma reetichetării una sau mai multe instrucţiuni ar urma să primească etichete mai mari decît limita admisibilă (9999), sistemul execută în mod automat comanda:

RENUMBER 1 STEP 1

Alte comenzi

Comanda SIZE

Are formatul: SIZE ↵. Este utilizată pentru obţinerea unor informaţii de la sistem. Determină scrierea la consolă a numărului de baiţi folosiţi de către program, precum şi a numărului de baiţi ce mai sînt încă disponibili. Scrierea se face în numere zecimale. *Exemplu:*

SIZE ↵
USED: 7200 BYTES
LEFT: 2300 BYTES

Comanda PAGE

Are formatul:
PAGE = n ↵
unde n este un întreg în gama: $1 \leqslant n \leqslant 132$.

Comanda stabileşte prin „n" limita lăţimii unei pagini de imprimare. În lipsa acestei comenzi sistemul consideră această lăţime ca fiind egală cu 72 de caractere. *Exemplu:*

PAGE = 120

Comanda TAB

Are formatul:

TAB=n ↵

unde n este un întreg cuprins în gama: $1 \leqslant n \leqslant$ dimensiunea paginii.

Efectul comenzii este împărţirea unei linii de imprimare în zone de lungime „n" caractere.

Parametrul „dimensiunea paginii" a fost fixat printr-o comandă PAGE anterioară sau dacă nu s-a dat o astfel de comandă, are valoarea 72.

În lipsa acestei comenzi pagina se consideră împărţită în zone de lungime egală cu 14 caractere.

Exemple:

a) Comenzile PAGE=120
TAB=12

determină împărţirea unei linii de imprimare în 10 zone a cîte 12 poziţii de caracter.

b) Comanda TAB=10 determină împărţirea liniei de imprimarea (presupusă de lungime 72 caractere) în 7 zone a cîte 10 caractere fiecare.

Comanda NEW

Are formatul: NEW. Determină ştergerea tuturor instrucţiunilor, variabilelor şi masivelor curente şi închide toate canalele deschise. Comanda se dă de obicei înainte de a introduce sau încărca un nou program. Ea poate fi utilizată şi ca instrucţiune de program în relaţie cu instrucţiunea ON, pentru prevenirea unei citiri neautorizate a programului.

Comanda BYE

Are formatul: BYE ↵. Marchează închiderea unei sesiuni la terminal, adică întrerupe interacţiunea dintre sistem şi utilizator. Întreruperea legăturii logice între terminal şi sistem este însoţită şi de alte efecte care depind de consolă de la care a fost editată comanda BYE. Într-o configuraţie cu acces multiplu există o consolă primară („master console"), de regulă consola operatorului de sistem şi mai multe console pentru utilizatori. La executarea unei comenzi BYE venită de la un terminal (teletype) se apelează o procedură de încheiere a sesiunii la terminal („sign — off procedure"), cu care prilej sistemul tipăreşte la consola terminalului data calendaristică şi timpul consumat la unitatea centrală şi dispozitivele de intrare/ieşire în cadrul sesiunii respective. În continuare terminalul devine inactiv.

În cazul executării comenzii BYE de la consola primară apar aceleaşi informaţii de mai sus şi în plus tipăreşte mesajul de interogare:

DIRECTORY SPECIFIER:

prin care se cere introducerea unui nume director de utilizator (vezi cap. 11).

Capitolul IV

ELEMENTELE LIMBAJULUI BASIC

4.1. INTRODUCERE

Limbajul BASIC fiind elaborat de specialişti care vorbesc limba engleză conţine cuvinte în această limbă. Acest amănunt nu prezintă însă importanţă dacă cuvintele şi instrucţiunile limbajului sînt interpretate drept notaţii pentru noţiunile utilizate în mod curent în redactarea unui proces de calcul în limba română.

În cele ce urmează textele scrise în BASIC vor fi redactate cu litere mari specificînd elemente ale limbajului şi cu litere mici, pentru explicarea lor.

Se obişnuieşte interpretarea [40] în sensul că noţiunile explicate aparţin unei limbi în timp ce explicaţiile aparţin altei limbi numită metalimbă. În cazul de faţă limba română joacă rolul de metalimbă, iar limbajul BASIC rolul de limbă.

BASIC, fiind un limbaj, are gramatică, vocabular şi semantică. Gramatica se referă la regulile sintactice pentru scrierea instrucţiunilor; vocabularul constă dintr-o mulţime de simboluri cu ajutorul cărora se realizează cuvinte; semantica conţine o serie de reguli cu privire la logica programului [37].

4.2. DEFINIŢII BASIC

Realizarea programelor pentru o clasă largă de probleme, atît algebrice cît şi de prelucrarea datelor, presupune cunoaşterea unui set de nouă reguli fundamentale, care sînt prezentate în cele ce urmează [42].

4.2.1. CARACTERE ALFANUMERICE

Mulţimea caracterelor acceptate este:

A. Cifre: 0, 1, 2, ..., 9

B. Litere: *A*, *B*, ..., *I*, ..., *X*

C. Caractere speciale: $*( ), . + − / =' > < " ↑ ; ∅ (∅ indică spaţiu liber).

4.2.2. ETICHETE

Orice instrucţiune conţine o etichetă formată din maxim 4 cifre:

Exemple: 1
24
319
1000

4.2.3. TEXTE

Orice şir de caractere introdus între ghilimele poate constitui un text

Exemple: "NUME STUDENT"
"PROGRAM APLICATIV"
"DATA 31/12/1975"

4.2.4. CONSTANTE

Orice şir de cifre cu sau fără semn, (numărul fără semne este considerat pozitiv) cu sau fără punct zecimal este o constantă.

Exemple: 21
−13.151
−.004
+78.19

4.2.5. VARIABILE

A. *Variabile simple*: orice literă, sau orice literă urmată de o cifră pot constitui nume de variabile.

Exemple: *A*, *X*, *V*
*A*1, *X*7, *V*4

B. *Variabile cu indici*. Numele variabilei respectă regula de la punctul A. Se admit variabile cu unu sau doi indici.

Indicele poate fi o constantă, o variabilă sau o expresie aritmetică, încadrată în paranteze.

Exemple: *A*(1), *B*(*I*), *C*(*K*+1)
A(1,5), *B*(*I*, *J*), *C*(3, *X*/*Y*)

C. *Variabila* şir poate fi orice şir de caractere alfanumerice. Nume de astfel de variabilă poate fi orice literă a alfabetului urmată de semnul dolarului

Exemple: *A*$, *B*$, *X*$

4.2.6. OPERATORI

Se utilizează următoarele simboluri pentru specificarea operaţiilor aritmetice:

Simbol	*Semnificaţie*
+	adunare
−	scădere
*	înmulţire
/	împărţire
↑	exponenţiere

4.2.7. EXPRESII

O expresie poate fi o constantă, o variabilă sau un şir de variabile şi constante legate prin operatori şi îndeplinind anumite reguli de sintaxă.

Exemple:

$A+10$
$-5+N9\uparrow X+100$
$X(I, J)\uparrow X(I, J)/10+.003$

4.2.8. COMPARAŢII

Se utilizează următoarele simboluri pentru specificarea comparaţiilor:

Simbol	*Semnificaţie*
=	egal cu
< >	inegal cu
>	mai mare decît
> =	mai mare sau egal cu
<	mai mic decît
< =	mai mic sau egal cu

4.2.9. DATE

Orice constantă poate constitui o dată

Exemple: .085
+194
− .004
−17.35

4.3. ELEMENTE DE GRAMATICĂ

4.3.1. FORMATUL INSTRUCŢIUNII

Orice instrucţiune scrisă în limbajul BASIC are următoarea structură:

n n n n MNEMONICA PARAMETRII

unde:

n n n — reprezintă eticheta; aceasta indică ordinea de compilare şi executare şi trebuie să fie unică (nu este permisă utilizarea aceleiaşi etichete la două instrucţiuni din program); dată

fiind funcţia ei, este util pentru programator să lucreze cu etichete consecutive care să difere prin mai mult decît o cifră, avînd astfel posibilitatea intercalării ulterioare a unor instrucţiuni;

MNEMONICA — indică tipul instrucţiunii care se va executa;

PARAMETRII — reprezintă informaţii definitorii pentru realizarea instrucţiunii şi sînt în general constante, variabile sau expresii.

Un exemplu de instrucţiune ar fi:

10 LET X=5

10 reprezintă eticheta, LET mnemonica şi X=5 parametrii.

4.3.2. SPAŢIEREA

Spaţiile libere nu sînt necesare în scrierea instrucţiunii. Este însă preferabilă scrierea utilizînd spaţiile libere, aceasta permiţînd o mai uşoară citire şi depanare a programelor.

Instrucţiunea din exemplul următor este validă din punct de vedere gramatical, dar citirea ei este dificilă:

10IF*K*=10GOTO100

Este evident mai convenabilă scrierea:

10 IF *K*=10 GO TO 100

4.4. COMENTARII

Într-un program scris în limbajul BASIC se pot introduce comentarii făcînd uz de instrucţiunea REM al cărei format este:

n n n n REM comentariu

Dacă textul care reprezintă comentariul depăşeşte capacitatea unei linii (72 de caractere) atunci trebuiesc utilizate mai multe instrucţiuni REM consecutive.

Exemple:

10 REM PROGRAM DE REZOLVARE A SISTEMELOR DE ECUAŢII
LINIARE
20 REM VARIABILA *D* REPREZINTĂ DETERMINANTUL
30 REM VARIABILELE *A*, *B*, *E*, *F* REPREZINTĂ
40 REM COEFICIENŢII

Instrucţiunea REM este o instrucţiune neexecutabilă, efectul ei fiind doar de tipărire a comentariului, ca parte a programului listat.

Este posibilă comanda unui salt în program, la eticheta instrucţiunii REM; aceasta fiind o instrucţiune neexecutabilă va fi ignorată, comanda fiind transferată primei instrucţiuni executabile întîlnită în secvenţă.

4.5. DATE

În limbajul BASIC pot fi prelucrate date numerice sau alfanumerice.

Datele numerice pot fi numere algebrice care nu depăşesc 6 cifre zecimale [65].

Pentru cazul numerelor care depăşesc 6 cifre zecimale se utilizează forma exponenţială simplă precizie (maxim 6 cifre urmate de litera $E\pm$exponent), sau dublă precizie (maxim 8 cifre urmate de litera $E\pm$exponent).

Exemple:

274	$-.5$	$-.004$	193.27
$1E+7$	$-1.534E-5$	$-.7E-8$	$18.39\,E+6$

Datele alfanumerice pot fi şiruri de caractere (cifre + litere) a căror lungime maximă este limitată la 132 de caractere.

Şirurile de caractere pot fi atribuite la nume de variabile şir prin instrucţiunile LET, INPUT, READ; se pot realiza comparaţii între două şiruri de caractere. Nu se pot executa asupra lor operaţii aritmetice.

Exemple: *AB37F*,

LET *A*$=DETA 5

4.6. INSTRUCŢIUNEA STOP

Formatul instrucţiunii este

n n n n STOP

Această instrucţiune este utilizată pentru a opri execuţia programului într-un punct dorit. Execuţia ei are drept urmare, pe lîngă oprirea programului, tipărirea la consolă a mesajului:

STOP n n n n

unde **n n n n** reprezintă eticheta instrucţiunii STOP din program.

Se pot utiliza într-un program mai multe instrucţiuni STOP, dacă acest lucru este necesar.

Exemple de utilizare a acestei instrucţiuni vor fi date în capitolul de APLICAŢII, după ce au fost prezentate toate instrucţiunile limbajului.

4.7. INSTRUCŢIUNEA END

Instrucţiunea END specifică sfîrşitul logic al programului şi are formatul:

n n n n END

unde, **n n n n** trebuie să fie cel mai mare număr de etichetă din programul respectiv.

Programul este terminat în momentul detectării instrucţiunii END.

Exerciţii

1. De ce în BASIC aceeaşi etichetă nu poate fi utilizată de mai multe ori?
2. Care este diferenţa între mnemonica şi parametrii unei instrucţiuni?
3. Cum se pot introduce comentariile într-un program scris în limbajul BASIC?
4. Ce restricţii se impun în prelucrarea şirurilor de caractere alfanumerice?
5. Care este ultima instrucţiune a programului?
6. Să se identifice care dintre următoarele nume de variabile sînt incorecte şi să se specifice de ce:

A	ART	DATA
Az	.5	AO
B1	8	MM
EC	X1$	MX3
7	X$	D2
4X	X$	D(7)

7. Care dintre următoarele constante BASIC sînt invalide şi de ce?

1453	.0003512
19.2741	$-3.123\ E-3$
− .7538412	$.174\ E-7$
−12491247	$-1249875\ E+9$
+17.359123	$.03714211\ E-5$

Capitolul V

INTRODUCEREA DATELOR

În scopul introducerii de la tastatura consolei, în memoria internă a calculatorului, a datelor necesare unui program, sînt utilizate instrucţiunile READ-DATA, INPUT, RESTORE; masivele de date sau şirurile de caractere, pentru a putea fi citite, sînt anterior declarate prin instrucţiunea neexecutabilă DIM [65].

5.1. INSTRUCŢIUNILE READ ŞI DATA

Instrucţiunea neexecutabilă DATA este utilizată pentru a furniza valori numerice sau alfanumerice variabilelor din program, valori ce vor fi citite prin execuţia instrucţiunii READ.

Formatul instrucţiunii DATA este:

n n n n DATA listă de constante

DATA — reprezintă mnemonica care specifică că următoarele informaţii reprezintă o secvenţă de date;

lista de constante — poate conţine una sau mai multe constante valide BASIC; elementele listei trebuie să fie separate prin virgule; şirurile de caractere alfanumerice trebuiesc încadrate în ghilimele.

Exemple:
```
 100 DATA 13
  10 DATA 11.5,−7,123.01,0.91
1000 DATA 1.03, „VALORILE PT. A, B, C“, 1, 9, 11
```

dacă lista de valori nu poate fi introdusă într-o singură instrucţiune DATA, se pot utiliza mai multe instrucţiuni consecutive, cu etichete ordonate crescător.

Exemplu:
```
100 DATA 1, 3, 5, 7, 9, 11, 13, 17, 19
101 DATA 2, 4, 6, 8, 10, 12, 14, 16
102 DATA.01, .02, .03, .04, 05.
```

De remarcat faptul că la sfîrşitul listei nu se utilizează virgula.

Valorile din lista instrucţiunilor DATA sînt memorate într-un singur bloc de date înainte de execuţia programului; ele sînt apoi atribuite variabilelor, în ordinea în care apar în DATA, la momentul execuţiei instrucţiunilor READ.

Instrucţiunea DATA, fiind neexecutabilă, poate fi plasată oriunde în program, se obişnuieşte însă a fi plasată la sfîrşitul programului, în felul acesta toate datele programului sînt grupate în mod unitar, iar o eventuală depanare a programului este mai uşor de realizat.

Formatul instrucţiunii READ este:

n n n n READ listă de variabile

READ — reprezintă mnemonica care indică tipul instrucţiunii
listă de variabile — poate conţine una sau mai multe variabile valide BASIC; elementele listei trebuie să fie separate prin virgule.

Executarea instrucţiunii READ determină alocarea valorilor din lista instrucţiunii DATA, variabilelor din lista instrucţiunii READ, în ordinea în care au fost declarate în DATA. De exemplu:

10 READ *X*, *Y*

100 DATA .005,113

Executarea instrucţiunii READ va determina alocarea valorii 0,005 variabilei *X* şi 113 variabilei *Y*.

Se consideră următorul exemplu

10 READ *A*

100 DATA 1, 3, 5, 7, 9, 11

Execuţia repetată a instrucţiunii READ va provoca alocarea, pe rînd, a valorilor din DATA, variabilei *A*. Acest lucru este posibil datorită existenţei unui contor care este incrementat pentru fiecare valoare citită din lista de date. O nouă încercare de execuţie a instrucţiunii READ, după ce ultima valoare din blocul de date a fost alocată, provoacă apariţia unei erori de "lipsă de date". Acelaşi bloc de date poate fi utilizat de mai multe ori făcînd apel la instrucţiunea RESTORE.

Variabilele din lista instrucţiunii READ trebuie să fie de acelaşi tip cu datele din lista instrucţiunii DATA. Încercarea de a atribui o valoare aritmetică unei variabile şir, de exemplu, va determina apariţia unui mesaj de eroare.

Exemplul 1

10 READ *N*$, *T*$, *A*

100 DATA "ION POPESCU"

101 DATA "NR. TELEFON", 330531

Variabilelor şir *N*$ şi *T*$ li se vor atribui şirurile de caractere ION POPESCU şi NR. TELEFON respectiv, iar variabilei *A* numărul de 6 cifre 33 05 31.

Exemplul 2

10 READ *X*, *Y*, *Z*

20 READ *A*

30 READ *M*(*I*)

100 DATA 1.5, –.3, –.4,7, 11.2, –18.4, .003

Vor fi alocate valorile 1.5, –0.3, –0,4, 7, variabilelor *X*, *Y*, *Z*, *A* respectiv şi valoarea 11.2 variabilei *M*(1) (pentru indicele *I*=1). Dacă se asigură reluarea instrucţiunii 30 pentru alte valori ale indicelui *I*, variabila *M*(*I*) va căpăta respectiv valorile –18.4 şi 0.003.

Exemplul 3

```
10 READ N$, F$
20 READ A$, B$, C$, D$, E$
30 READ N1, N2, N3, N4, N5
100 DATA "DAN POPA", "CIBERNETICA"
101 DATA "ECONOMIE", "SISTEME", "SPORT", "CONTABILITATE"
102 DATA "MATEMATICI", 8, 10, 10, 7, 9
```

Executarea acestor instrucţiuni permite introducerea în memoria calculatorului a unor date privitoare la un student, utilizînd variabilele:

N$ — numele studentului;
F$ — facultatea la care este înscris;
A$ ÷ *E*$ — titlul cursurilor la care susţine examene;
*N*1 ÷ *N*5 — notele obţinute la examene;

Evident că utilizarea variabilelor respective face posibilă introducerea datelor pentru orice student şi orice facultate schimbînd numai elementele din lista instrucţiunilor neexecutabile DATA.

5.2. INSTRUCŢIUNEA RESTORE

Formatul instrucţiunii este:

n n n n RESTORE

Instrucţiunea permite utilizarea unui bloc de date de mai multe ori, executarea ei avînd drept efect repoziţionarea contorului aferent blocului de date la prima valoare. Instrucţiunea READ ce urmează unei instrucţiuni RESTORE va citi blocul de date respectiv începînd cu primul element.

Exemplul următor ilustrează modul de utilizare a acestei instrucţiuni:

```
10 READ A, B, C
20 RESTORE
30 READ X, Y, Z
.
.
.
100 RESTORE
110 READ A1, A2, A3
1000 DATA -.01, -0.02, -.03
```

Valorile −0.01, −0.02, −0.03 vor fi alocate pentru variabilele *A*, *B*, *C* apoi pentru X, *Y*, *Z* şi în final pentru *A*1, *A*2, *A*3.

5.3. INSTRUCŢIUNEA INPUT

Instrucţiunea INPUT reprezintă o facilitate de introducere a datelor de la tastatura consolei în timpul execuţiei programului. Este utilizată în special în cazul în care conversaţia programator-calculator este necesară pentru rezolvarea problemei.

Formatul instrucţiunii este:

n n n n INPUT listă de variabile

INPUT — mnemonica care indică tipul instrucţiunii;
lista de variabile — poate conţine unul sau mai multe nume de variabile valide BASIC; elementele listei trebuie să fie separate prin virgule.

În secvenţa logică a programului instrucţiunea INPUT va fi introdusă acolo unde sînt necesare datele; în timpul rulării programului, executarea ei, provoacă trecerea programului în starea de aşteptare şi tipărirea la consolă a caracterului „?"; după aceasta operatorul poate introduce datele de la tastatura consolei; datele vor fi transferate variabilelor specificate în instrucţiunea INPUT după apăsarea tastei RETURN.

Elementele listei de date pot fi separate prin virgule sau prin apăsarea tastei RETURN (apăsarea tastei RETURN în aceste condiţii nu provoacă întoarcerea carului ci semnalează sistemului că data respectivă a fost terminată); în caz de nesatisfacerea listei din INPUT X la consolă apare caracterul „?", iar operatorul poate continua introducerea datelor.

De exemplu:

10 INPUT *A*, *B*, *C*
20 INPUT *A*$

Executarea acestor două instrucţiuni determină apariţia la consolă a semnului „?" după care operatorul poate tipări datele:

?1.1↵?1.2↵? 1 3↵
? PROGRAM

Semnul ↵ specifică apăsarea tastei RETURN.

Aceleaşi date puteau fi introduse şi utilizînd ca separator virgula:

? 1.1, 1.2, 1 3
? PROGRAM

În lista de date, şirurile de caractere pot fi opţional introduse între ghilimele.

Într-o secvenţă de instrucţiuni INPUT, lista de variabile poate fi terminată prin „;"; acest lucru va determina tipărirea datelor pentru următoarea instrucţiune INPUT pe aceeaşi linie cu datele precedentei.

Exemplu:

10 INPUT *A*$, *E*$;
20 INPUT *N*$, *E*

În urma execuţiei acestor două instrucţiuni operatorul va introduce datele:

? "STUDENT", EXAMEN, "CARMEN", 10

Şirurile de caractere STUDENT şi CARMEN au fost încadrate în ghilimele în timp ce EXAMEN nu.

Dacă după instrucţiunea 10 nu ar fi fost utilizat caracterul „;" atunci datele ar fi fost introduse pe două linii distincte:

10 INPUT *A*$, *E*$
20 INPUT *N*$, *E*
? STUDENT, EXAMEN
? "CARMEN", 10

Lista de variabile din INPUT trebuie să concorde cu lista de date introduse de la consolă atît ca tip cît şi ca număr; în caz contrar este provocată apariţia unui mesaj de eroare.

Exemplu:

10 INPUT *X*\$, *X*, *A*, *B*, *C*
? 5.3, "VALOARE", 4, 1

Introducerea acestor date este incorectă atît ca număr cît şi ca tip de valori; sînt 4 valori pentru 5 variabile iar variabilei şir i se alocă o valoare aritmetică şi invers.

Utilizarea instrucţiunii INPUT prezintă un inconvenient prin aceea că operatorul, la momentul introducerii datelor, nu cunoaşte lista de variabile, sistemul tipărind numai semnul întrebării. Acest lucru poate fi compensat prin utilizarea corespunzătoare a unei instrucţiuni PRINT, (această instrucţiune determină tipărirea listei respective de variabile; este descrisă în cap. IX) aşa cum se observă pe exemplul următor:

20 PRINT "VALORILE PENTRU X, Y, *Z*"
30 INPUT *X*, *Y*, *Z*

Executarea instrucţiunilor 20 şi 30 determină tipărirea următoarei secvenţe:
VALORILE PENTRU *X*, *Y*, *Z*

? 11.3, 17, —.07

Instrucţiunea PRINT poate fi utilizată, pentru verificare, şi după INPUT.

Exemplu:

10 INPUT *A*\$. *B*\$, *A*
20 PRINT *A*\$, *B*\$, *A*
? 7←MARINESCU, SISM←TEME, 7

Operatorul a făcut două greşeli în timpul introducerii datelor pe care le-a corectat prin apăsare pe tasta RUBOUT (se specifică prin săgeată).

Instrucţiunea PRINT provoacă tipărirea următoarei linii:

MARINESCU SISTEME 7

5.4. DIMENSIONAREA MASIVELOR DE DATE

Limbajul BASIC permite prelucrarea şirurilor de caractere şi a masivelor de date cu una (liste) sau două (tabele) dimensiuni [65]; variabilele cu indici aferente pot fi utilizate fără rezervare specială de memorie dacă dimensiunea maximă pentru fiecare indice nu depăşeşte 10; în caz contrar dimensionarea masivelor sau a şirurilor de caractere este absolut necesară.

Instrucţiunea neexecutabilă **DIM** este utilizată pentru rezervarea de memorie necesară masivelor. Formatul ei este:

n n n n DIM variabilă (întreg), variabilă (întreg, întreg),...

DIM — mnemonica care specifică tipul instrucţiunii (rezervare de memorie);
variabila — poate fi orice nume de variabilă cu indici sau variabilă şir;
întreg — specifică tipul masivului (una sau două dimensiuni) şi dimensiunea maximă; pentru cazul şirurilor de caractere indică numărul maxim de caractere din şir.

— Limita inferioară a dimensiunii este întotdeauna 0 (zero);
— În cazul masivului bidimensional cele două limite superioare trebuie separate prin virgulă;
— Limita superioară a dimensiunii poate fi inclusă în paranteze ronde sau drepte;
— Elementele listei DIM trebuiesc separate prin virgule.

Exemplu:

10 DIM *A*(15), *B*(5,7), *X*\$(20), *M*(13,13)
Prin această instrucţiune au fost declarate:
A o listă de 16 elemente
B un tablou de 6×8 elemente
X\$ un şir de 20 de caractere
M un tablou de 14×14 elemente

E x e r c i ţ i i

1. Ce valoare va fi alocată variabilei *W* după executarea următoarelor declaraţii BASIC:
10 DATA 2, 3, 4, 7, 11, 12, 13
20 DATA 15, 7, 19.3.5.2
30 READ *A*, *B*, *C*, *D*
40 READ *W*, *X*, *Y*, *Z*
2. Este corectă următoarea secvenţă BASIC?
10 READ *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*
20 READ X. *Y*, *Z*
30 DATA 1, 3, 5, 2, 4
3. Este corectă următoarea secvenţă BASIC?
10 READ *A*\$, *B*\$, *X*, *Z*\$
20 READ *A*(*I*, *Y*)
30 DATA "ELEMENTE", 21, 17, 1.5
40 DATA "BASIC", 1, 1.1,1.2,1.3

Capitolul VI

OPERAŢII ARITMETICE

În acest capitol ne vom ocupa de modul cum pot fi realizate diversele operaţii matematice în limbajul BASIC [42]. Expresiile aritmetice apar în forme foarte diferite, dar rezultatul final al execuţiei unei instrucţiuni aritmetice constă în alocarea unei constante, variabile sau expresii unei variabile specificate. Această operaţie este realizată în BASIC de instrucţiunea LET.

Instrucţiunea LET, a cărei execuţie determină realizarea unor calcule matematice şi alocarea rezultatului unei variabile, are următorul format:

n n n n LET variabilă = expresie

LET — mnemonica care indică tipul instrucţiunii
variabilă — poate fi orice nume valid de variabilă BASIC
= — indică faptul că rezultatul calculelor efectuate asupra expresiei trebuie alocat variabilei din stînga
expresie — poate fi orice constantă, variabilă sau expresie aritmetică validă BASIC

— mnemonica LET este opţională
— expresiile construite cu şiruri de caractere trebuiesc alocate variabilelor şir, iar expresiile aritmetice variabilelor aritmetice.

În cele ce urmează se dau cîteva exemple de instrucţiuni LET corecte:

```
 10 LET N=100
 20 LET A3=K
 30 LET A=3.14+M1
 45 LET D=B↑2-4*A*C
100 W=W+X↑(1/2)+(Z-A)/B
115 LET A$=„NOTA“
```

6.1. ADUNAREA ŞI SCĂDEREA

Considerăm două variabile *X* şi *Y* ale căror valori trebuiesc adunate, iar rezultatul atribuit fie unei alte variabile *Z* fie uneia dintre ele. Această operaţie poate fi realizată astfel:

```
10 LET X=11.25
20 LET Y=15.25
30 LET Z=X+Y
```

Dacă se reprezintă conţinutul locaţiilor de memorie ale variabilelor *X*, *Y*, *Z* atunci acestea arată astfel:

	X	*Y*	*Z*
Înainte de adunare	11.25	15.25	0
După adunare	11.25	15.25	26.50

Instrucţiunea 30 ar putea fi:

30 LET $X=X+Y$

În aceste condiţii rezultatul sumei va fi atribuit variabilei X, iar conţinutul locaţiilor va fi:

	X	Y	Z
Înainte de adunare	11.25	15.25	—
După adunare	26.50	15.25	—

Expresiile de tipul $A=B$ nu trebuiesc confundate cu corespondentul lor din algebră; sensul lor este de atribuire a conţinutului locaţiei B, locaţiei A. Este evident că în algebră o expresie de forma $X=X+Y$ nu poate avea sens decît pentru $Y=0$.

Cele specificate pentru adunare sînt valabile şi pentru scădere, astfel încît vom da numai un exemplu:

```
10 LET X=14.3
20 LET Y=0.1
30 LET A=X-Y
```

Conţinutul locaţiilor de memorie va fi:

	X	Y	A
Înainte de scădere	14.3	0.1	—
După scădere	14.3	0.1	14.2

```
10 LET X=14.3
20 LET Y=-.1
30 LET X=X-Y
```

Conţinutul locaţiilor de memorie va fi:

	X	Y	A
Înainte de scădere	14.3	−0.1	—
După scădere	14.4	−0.1	—

6.2. ÎNMULŢIREA ŞI ÎMPĂRŢIREA

Două constante sau variabile BASIC pot fi înmulţite utilizînd instrucţiunea LET, aşa cum rezultă din următorul exemplu:

```
10 LET A=11
20 LET B=2.01
30 LET A1=A*B
```

Conţinutul locaţiilor va fi:

	A	B	$A1$
Înainte de înmulţire	11	2.01	0
După înmulţire	11	2.01	22.11

Valoarea 11 este alocată variabilei A, valoarea 2.01 este alocată variabilei B, iar produsul calculat dintre A şi B este alocat variabilei $A1$.

Similar se poate realiza şi împărţirea a două constante sau variabile:

```
35 LET X=100
45 LET A=15
100 LET W=X/A
```

Instrucţiunea 35 alocă variabilei X valoarea 100, instrucţiunea 45 alocă variabilei A valoarea 15, iar instrucţiunea 100 realizează împărţirea lui X la A şi atribuie rezultatul variabilei W.

Este de remarcat faptul că deşi variabilele X şi A au valori întregi, împărţirea determinînd apariţia părţii fracţionare, aceasta este automat reţinută; în condiţiile apariţiei fracţiilor iraţionale este asigurată precizia maximă admisibilă în BASIC (problemă tratată în capitolul IV).

Conţinutul locaţiilor de memorie va fi:

	X	A	W
Înainte de împărţire	100	15	0
După împărţire	100	15	6.66666

6.3. IERARHIA DE EXECUŢIE A OPERAŢIILOR

Apariţia într-o expresie, a mai multor operatori aritmetici impune stabilirea unei ordini de execuţie a acestora. Ordinea de priorităţi este următoarea [40]:

Paranteze	Cea mai înaltă prioritate
↑	
* /	↓
+ −	Cea mai slabă prioritate

În aceste condiţii o expresie de forma:

```
100 LET X=A+B/C
```

va fi evaluată astfel: întîi împărţirea lui B la C apoi rezultatul adunat la A şi rezultatul final atribuit lui X.

Exemplul 1

```
5 LET V=1.1
10 LET A=3.14+X
20 LET V=V+A ↑ 1.2
30 LET M=V+A/X+17.3
```

Exemplul 2

```
10 LET A=K ↑ 2*X ↑ 3/4.1+R/B
```

Expresia corespunde următoarei expresii aritmetice:

$$a=\frac{K^2\cdot X^3}{4{\cdot}1}+\frac{R}{B}$$

6.4. PARANTEZE

Ordinea de priorităţi prestabilită în executarea operaţiilor aritmetice poate fi schimbată prin utilizarea convenabilă a parantezelor. Aşa cum rezultă din 6.3 parantezele au cel mai înalt nivel de prioritate; în aceste condiţii o expresie care dorim să fie prima evaluată va fi introdusă în paranteze.

În exemplul următor se doreşte realizarea următoarei expresii algebrice

$$x=\frac{a+b}{c\cdot d}$$

Scrierea unei instrucţiuni:

10 LET $X=A+B/C*D$

va determina calculul expresiei:

$$x=a+\frac{b}{c}\cdot d$$

ceea ce nu corespunde cu expresia dorită.

Utilizînd parantezele instrucţiunea devine:

10 LET $X=(A+B)/(C*D)$

În aceste condiţii întîi se va calcula suma $A+B$, apoi produsul $C*D$ şi în final primul rezultat va fi împărţit la cel de-al doilea, cîtul fiind atribuit variabilei X.

O expresie care conţine paranteze închise în paranteze este evaluată pornind de la cea mai internă către cea externă.

Exemplu:

10 LET $P=(((A+B)/2)\uparrow 1/2)+F$

Instrucţiunea realizează calculul următoarei expresii algebrice:

$$p=\sqrt{\frac{a+b}{2}}+f$$

Se evaluează în ordinea următoare:

a) $A+B$ b) $\frac{A+B}{2}$ c) $\left(\frac{A+B}{2}\right)^{1/2}$ d) $\left(\frac{A+B}{2}\right)^{1/2}+F$

6.5. OPERAŢII ALFANUMERICE

Asupra şirurilor de caractere pot fi executate numai operaţii de atribuire: atribuirea unui şir de caractere unei variabile şir, sau a unei variabile şir altei variabile şir.

Exemple:

5 LET A$=„NUMELE STUDENTULUI"
10 LET M$=„CALCULUL MEDIEI"
15 LET X$=A$
20 LET P$=M$

Prin instrucţiunile 15 şi 20 şirurile de caractere aferente variabilelor A$ şi M$ au fost atribuite variabilelor X$ şi P$ respectiv.

În exemplele următoare sînt prezentate cîteva declaraţii, asupra şirurilor de caractere, invalide:

10 LET A$=B
20 LET B$=11.3
30 LET X=A$
40 LET M=X*Y+A$

În instrucţiunea 10 se încearcă atribuirea unei variabile aritmetice variabilei şir iar în instrucţiunea 30 atribuirea unui şir de caractere unei variabile aritmetice. Instrucţiunea 20 atribuie o constantă unei variabile şir, iar instrucţiunea 40 conţine o expresie aritmetică alcătuită cu variabile aritmetice şi variabile şir. Toate aceste operaţii sînt nepermise în limbajul BASIC şi vor provoca apariţia unui mesaj de eroare.

6.6. APLICAŢII

A. Să se rezolve sistemul [12]:

$$\begin{cases} ax+by=e \\ cx+dy=f \end{cases}$$

Pentru rezolvarea acestui sistem liniar de 2 ecuaţii cu două necunoscute s-a realizat următorul program BASIC.

```
10 REM PROGRAM SISTEM
20 REM VARIABILELE A, B, C, D REPREZINTĂ COEFICIENŢII
30 REM VARIABILELE E, F — TERMENII LIBERI
40 REM VARIABILA M — DETERMINANTUL SISTEMULUI
50 READ A, B, C, D
60 LET M=A*D-B*C
70 READ E, F
80 LET X=(E*D-B*F)/M
90 LET Y=(A*F-E*C)/M
100 DATA 5, 7, 13, 19, 10, 15
```

După cum se observă variabilele au fost definite la începutul programului prin comentarii (instrucţiunile 20-40); instrucţiunile 50 şi 70 realizează citirea coeficienţilor sistemului din blocul de date indicat în lista instrucţiunii 100; instrucţiunea 60 determină calculul determinantului sistemului, iar instrucţiunile 80, 90 calculează necunoscutele x şi y.

B. Într-un magazin 3 articole nu au fost vîndute în sezon; ca urmare a trecerii sezonului acestea au fost ieftinite; nefiind vîndute nici în aceste condiţii s-a operat o nouă ieftinire asupra lor. Preţurile iniţiale precum şi procentele de ieftinire fiind date în tabelul de mai jos se cere să se calculeze ultimul preţ pentru fiecare articol [60].

	Preţ iniţial	*Procent de ieftinire 1*	*Procent de ieftinire 2*
Articolul 1	840	20%	10%
Articolul 2	2400	25%	5%
Articolul 3	170	20%	20%

Programul BASIC care realizează calculul preţurilor după cele două ieftiniri este:

```
10 REM PROGRAM PREŢURI
20 REM VARIABILELE P1, P2, P3 — PREŢURI
30 REM VARIABILELE I1, I2, I3 — PROCENT DE IEFTINIRE 1
40 REM VARIABILELE K1, K2, K3 — PROCENT DE IEFTINIRE 2
50 READ P1, I1, K1
60 LET P1=P1*(1-I1/100)
70 LET P1=P1*(1-K1/100)
80 REM P1 — PREŢUL ACTUAL
90 READ P2, I2, K2
100 LET P2=P2*(1-I2/100)
110 LET P2=P2*(1-K2/100)
120 REM P2 — PREŢUL ACTUAL
130 READ P3, I3, K3
```

```
140 LET P3=P3*(1-I3/100)
150 LET P3=P3*(1-K3/100)
160 REM P3 - PREŢUL ACTUAL
170 DATA 840, 20, 10
180 DATA 2400, 25, 5
190 DATA 170, 20, 20
200 END
```

Variabilele au fost definite în program prin comentarii, iar preţul după cele două ieftiniri a fost calculat utilizînd de două ori consecutiv instrucţiunea LET; variabilele *P*1, *P*2, *P*3 au la sfîrşitul execuţiei programului valoarea preţului actualizat, deci conţinutul iniţial al locaţiei de memorie a fost pierdut.

Programul putea fi conceput într-o manieră mai compactă prin utilizarea instrucţiunilor de control. Propunem cititorului să încerce acest lucru după parcurgerea capitolului VII.

Exerciţii

1. Să se scrie o serie de instrucţiuni LET pentru alocarea următoarelor valori, variabilelor asociate:

	Variabile	date
a.	*N*	000.00
b.	*A*	1.1
c.	*M*2	17.113
d.	*A*$	DAN POPA
e.	*Z*	.000007
f.	*F*	10023417114
g.	*E*$	EXAMEN

2. Ce erori apar în următoarele instrucţiuni

```
10 LET A=18-6
20 LET BM=9.47
30 LET X$=11.3
40 LET A=C$+18.5
50 LET N$=A$+F$
60 LET P=(P+7)**2-A3/15)
```

3. Ce valoare va fi alocată variabilei *Z* în urma execuţiei următorului program:

```
1 DATA 1, 2, 3. 4, 6
2 DATA 7.8
3 READ A, B, X
4 READ Y, C, D
5 LET Z=X+Y
6 END
```

4. Care este ordinea de execuţie a operaţiilor în următoarele instrucţiuni:

```
10 LET A=A+B+C
20 LET B=(A+B+X*Y)*Z
25 LET C=A*B+C↑2
30 LET D=((A+B)+X*Y)*2
```

```
35 LET E=X+(Y*Z)
40 LET F=X*Y+A/B
45 LET G=A+B+C*D+X↑3
50 LET H=A/B/(C*D↑2)
```

5. Ce instrucţiuni BASIC sînt necesare pentru realizarea următoarelor expresii algebrice:

a) $x=a\cdot b$

b) $a=x+y-z-t$

c) $r=\frac{a+b}{c+d}$

d) $r=a+\frac{b}{c}+d$

e) $k=\sqrt{\frac{a+b}{c}}$

f) $y=\sqrt{x^n}$

g) $v=\frac{x+k/n}{m}$

h) $u=\frac{a+b}{x\cdot y}$

i) $t=\pi r\sqrt{r^2+h^2}$

j) $e=a^{nj}$

k) $z=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

l) $z=-a+b\cdot\frac{c}{d}+\left(\frac{e\cdot f}{g}+h\right)k$

m) $t=\frac{a}{\sqrt{(b+c)^2-m^3}}+\sqrt[5]{x}$

6. Se consideră următorul program; care este rezultatul după executarea fiecărei instrucţiuni LET:

```
10 LET A=1.0
20 LET B=2.0
30 LET N=4.0
40 LET X=A+B
50 LET Y=A/B
60 LET Z=A*B
70 LET W=X+Z-Y
80 LET X=X*Z↑B/N
90 LET C=Z↑B↑2
100 LET D=(((A*2*B)/N)↑3)-1
110 LET E=-N
120 LET N=-K*(-N)
```

7. În tabelul de mai jos sînt indicate numărul de ore prestate de 4 muncitori, precum şi retribuţia pe oră şi taxele respective. Să se realizeze un program BASIC pentru calculul sumei ce revine fiecărui muncitor.

Muncitor	Număr de ore	lei/oră	taxe
1	40	8.50	14%
2	50	7.25	10%
3	35	13.00	20%
4	40	8.00	13%

8. Scrieţi un program pentru calculul variabilei Z dată de expresia:

$$z=(a+b)^2-\left(\frac{x+y}{c}\right)^3\cdot a^{2b}+\frac{1}{a^2}$$

Introduceţi valorile $a=3$, $b=9$, $x=7$, $y=6$, $c=5$ la momentul execuţiei programului.

Capitolul VII

INSTRUCŢIUNI DE CONTROL

7.1. INTRODUCERE

Limbajul BASIC cuprinde un set de instrucţiuni de control care face limbajul elastic şi eficient.

Scopul acestei categorii de instrucţiuni este de a permite programatorului comanda nesecvenţială a executării unui program.

Instrucţiunile de control se pot clasifica în trei categorii:
— instrucţiuni de salt necondiţionat;
— instrucţiuni de salt condiţionat;
— instrucţiuni pentru descrierea ciclurilor.

7.2. INSTRUCŢIUNI DE TRANSFER NECONDIŢIONAT

Instrucţiunea de transfer necondiţionat utilizată în limbajul BASIC este **GOTO**.

Formatul instrucţiunii este

n n n n GOTO etichetă

Executarea acestei instrucţiuni determină transferul controlului la instrucţiunea cu eticheta menţionată în enunţ, dacă aceasta este o instrucţiune executabilă, iar dacă instrucţiunea respectivă este neexecutabilă, la prima instrucţiune executabilă care îi urmează.

Exemplul 1

Să se calculeze expresia:

$$E=x^2+y^2-2xy\cdot\cos z$$

pentru cîte n valori ale variabilelor x, y, z.

```
10 READ X, Y, Z
20 LET E=SQR(X ↑ 2+Y ↑ 2−2*X*Y*COS (Z))
30 GOTO 10
100 DATA 21, −7,5, −9, 11, 10, 73, −18, 15
```

Valorile pentru **X, Y, Z** vor fi citite din blocul de date şi se va calcula expresia E; operaţia se repetă pînă la terminarea setului de date din DATA.

Exemplul 2

```
10 DATA 5, .4,6.9, 7.684
20 READ X1, X2, Y1, Y2
30 LET X=X1+X2
```

```
40 GOTO 100
50 LET Y=Y1+Y2
60 GOTO 200
100 PRINT X
110 GOTO 50
200 PRINT Y
300 END
```

Programul realizează respectiv calculul și tipărirea lui $x=x_1+x_2$ și $y=y_1+y_2$

Exemplul 3

```
10 DATA 3, 4, 5
20 READ A, B, C
30 LET D=A+B-C
40 LET A=A+1
50 GOTO 30
60 END
```

Programul calculează expresia $d=a+b-c$, variabila a luînd pe rînd valorile 3, 4, 5 .

7.3. INSTRUCŢIUNI DE TRANSFER CONDIŢIONAT

Limbajul BASIC conţine două instrucţiuni de salt-condiţionat: **IF** şi **ON**.

Formatul instrucţiunii IF

n n n n IF expresie relaţie expresie {THEN | GOTO} etichetă

IF — reprezintă mnemonica care indică tipul instrucţiunii;
expresie — constă dintr-un nume de variabilă, o constantă sau o expresie aritmetică care prin evaluare determină o singură valoare;
relaţie — este un operator de comparaţie între cele două expresii;
etichetă — indică numărul instrucţiunii la care se transferă controlul programului, în cazul în care relaţia dintre expresii este adevărată

Operatorii de comparaţie utilizaţi sînt:

= egal cu
> mai mare decît
< mai mic decît
>= mai mare sau egal cu
<= mai mic sau egal cu
<> neegal cu

Sintaxa şi semantica instrucţiunii IF poate fi ilustrată prin următoarele exemple:

```
10 IF X=10 THEN 100
```

Dacă variabila x este egală cu 10 atunci controlul programului este trecut la instrucţiunea cu eticheta 100; în caz contrar programul se desfăşoară secvenţial.

```
10 IF X<>Y THEN 199
```

Dacă $x \neq y$ controlul este transferat la instrucţiunea cu eticheta 199.

```
10 IF X(99)<(A*B) THEN 800.
```

Dacă elementul x_{99} al vectorului x este mai mic decît produsul $a \cdot b$ atunci, în program are loc un salt la instrucţiunea 800.

10 *IFX*(I)∗(1/((1+*Y*(*I*))↑*N*(*I*)))>*X*(*I*+1)∗(1/((1+*Y*(*I*+1)↑*N*(*I*+1)))THEN111

Dacă este satisfăcută inegalitatea:

$$x(i)\,\frac{1}{(1+y(i))^{n(i)}} > x\,(i+1)\,\frac{1}{(1+y\,(i+1))^{\,n(i+1)}}$$

atunci se execută în program un salt la instrucţiunea cu eticheta 111.

Este de remarcat că într-o instrucţiune IF este permisă o singură comparaţie; dacă programul necesită mai multe comparaţii, acest lucru poate fi realizat prin utilizarea succesivă a mai multor instrucţiuni IF. Acest procedeu este ilustrat foarte clar prin următorul exemplu:

Exemplul 1

Să se sorteze, dintr-o mulţime de numere, acelea care sînt cuprinse între 20 şi 30.

Rezolvarea logică a problemei este prezentată în diagrama din fig. 7.1.

N — variabilă ce reprezintă numărul
C — variabilă contor
T — variabilă ce reprezintă numărul de numere

Fiecare număr este citit, testat dacă este mai mic decît 30, mai mare decît 20 şi dacă satisface ambele condiţii este tipărit; după citirea tuturor numerelor (*C*=*T*) programul se opreşte.

Programul aferent este:

```
10 READ T
20 LET C=1
30 READ N
40 IF N>30 THEN 30
50 IF N<20 THEN 30
60 LET C=C+1
70 PRINT N
80 IF C<=T THEN 30
90 DATA 10, -5, 15, 17, 19, 29, 112,
   37.3,8, 35, 7
```

Exemplul 2

Se consideră o mulţime de 100 numere; să se calculeze media lor aritmetică, şi suma abaterilor patratice ale fiecărui număr faţă de această medie.

Soluţia aleasă este cea prezentată în diagrama logică din fig. 7.2.

N(*I*) — vector cu 100 de componente care reprezintă numerele
S — variabilă ce reprezintă suma numerelor
*S*1 — variabilă ce reprezintă suma abaterilor patratice
M — variabilă ce reprezintă media numerelor

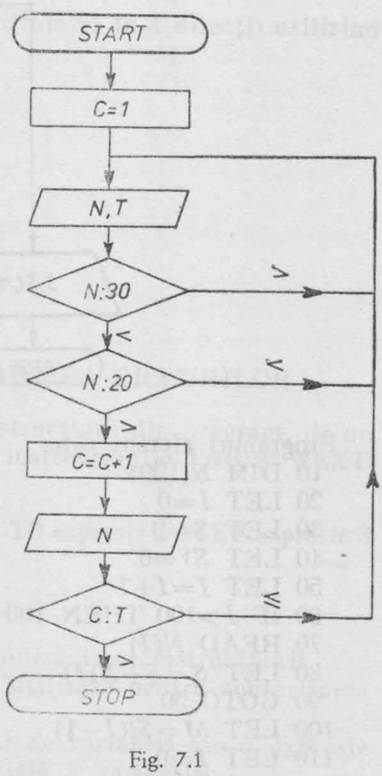


Fig. 7.1

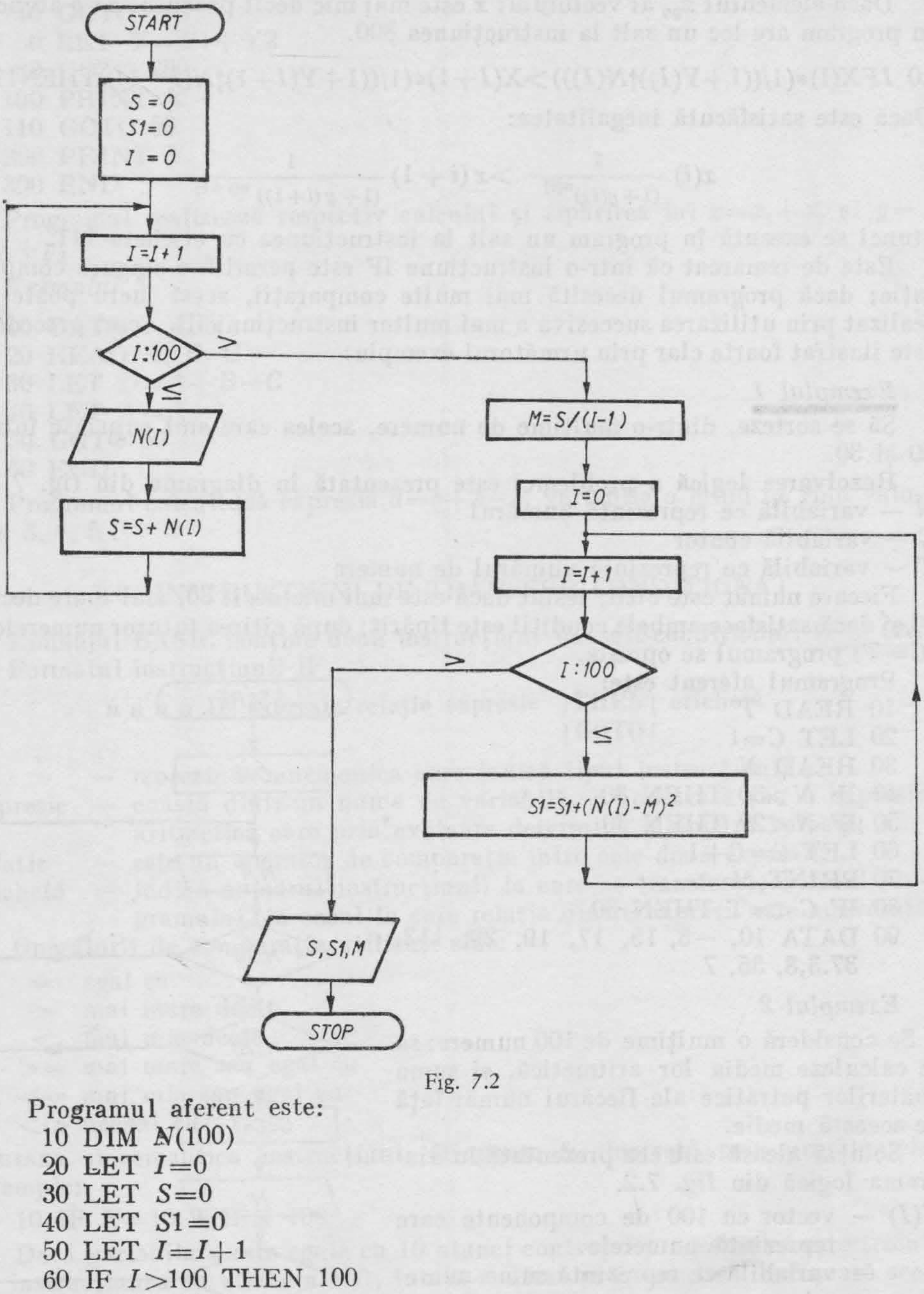


Fig. 7.2

Programul aferent este:

```
10 DIM N(100)
20 LET I=0
30 LET S=0
40 LET S1=0
50 LET I=I+1
60 IF I>100 THEN 100
70 READ N(I)
80 LET S=S+N(I)
90 GOTO 50
100 LET M=S/(I-1)
110 LET I=0
```

```
120 LET I=I+1
130 IF I>100 THEN 160
140 LET S1=S1+(N(I)-M) 2
150 GOTO 120
160 PRINT S, S1, M
170 DATA -7, 13, 12.1, 9.13, .21 ... , 8.25
180 END
```

Formatul instrucţiunii **ON**.

n n n n ON expresie GOTO listă de etichete

ON — mnemonica care indică tipul instrucţiunii

expresie — constă dintr-o singură variabilă, sau o expresie aritmetică care reprezintă un întreg; valoarea acestei expresii este testată.

listă de etichete — constă dintr-un număr de etichete de instrucţiuni către care se transferă controlul; dacă expresia capătă valoarea 1 atunci saltul se execută la prima etichetă din listă, dacă valoarea este 2, la a 2-a etichetă, etc.

Utilizarea acestei instrucţiuni este ilustrată în următoarele exemple:

```
10 ON M GOTO 100, 150, 200, 315.
```

În funcţie de valoarea (1, 2, 3, 4) pe care o ia variabila *M*, controlul este transferat respectiv la instrucţiunea cu eticheta 100, 150, 200, 315

```
10 ON(M-9)/10 GOTO 100, 150, 200, 315
```

Programul următor realizează transferul controlului în 4 direcţii utilizînd instrucţiunea ON.

```
10 READ M, I, X
20 LET X=X+1
30 LET I=I+1
40 IF X<=M GOTO 20
50 PRINT X, M, I
60 ON I GOTO 20, 30, 40, 70
70 STOP
80 DATA 10,0,6
90 END
```

7.4. INSTRUCŢIUNI PENTRU DESCRIEREA CICLURILOR

Pentru a permite repetarea unui grup de instrucţiuni din program, de un număr dorit de ori, limbajul BASIC utilizează instrucţiunile **FOR** şi **NEXT**.

Formatul acestor instrucţiuni este:

n n n n FOR variabilă = expresie 1 TO expresie 2 STEP expresie 3

•
•
•

n n n n NEXT variabilă

FOR — reprezintă mnemonica care indică tipul instrucţiunii

variabilă — reprezintă nume de variabilă utilizată pentru contorizarea numărului de iteraţii;

expresie 1 — poate fi o constantă, un nume de variabilă sau o expresie care reprezintă valoarea iniţială a variabilei;

expresie 2 — poate fi o constantă, un nume de variabilă sau o expresie care reprezintă valoarea finală a variabilei;

expresie 3 — poate fi o constantă, un nume de variabilă sau o expresie și indică cantitatea cu care este incrementată variabilă la fiecare ciclu; cînd aceasta lipsește incrementarea se face cu 1;

NEXT — indică sfîrșitul buclei.

Grupul de instrucțiuni cuprinse între FOR și NEXT va fi executat de un număr de ori, pînă cînd vor fi îndeplinite condițiile instrucțiunii FOR, apoi controlul va fi trecut în secvență, primei instrucțiuni care urmează instrucțiunii NEXT.

Exemplul 1

```
10 LET X=0
20 FOR I=1 TO 100
30 LET S=S+I
40 NEXT I
50 PRINT S
60 END
```

Programul calculează suma primelor 100 de numere naturale și apoi o tipărește.

Exemplul 2

```
 5 LET S=0
10 FOR I=1 TO 10
20 READ X(I)
30 LET S=S+X(I)
40 NEXT I
50 DIM X(10)
60 DATA −3,7.2, −.4, −11, .23, 17, 11.5, 2, 14, 9.3
70 END
```

Programul citește pe rînd elementele vectorului x și face suma lor.

Exemplul 3

```
10 FOR I=1 TO 3
20 FOR J=1 TO 5
30 READ A(I, J)
40 NEXT J
50 NEXT I
60 DATA 1, 0, 0, 1, 2, 1, 3, 4, 0, 1, 0, 2, 1, 3, 0
70 END
```

Programul realizează citirea pe linii a matricei

$$A=\begin{bmatrix} 1 & 0 & 0 & 1 & 2 \\ 1 & 3 & 4 & 0 & 1 \\ 0 & 2 & 1 & 3 & 0 \end{bmatrix}$$

Exemplul 4

Să se afle numerele formate din 3 cifre, a căror valoare este egală cu suma cuburilor celor 3 cifre.

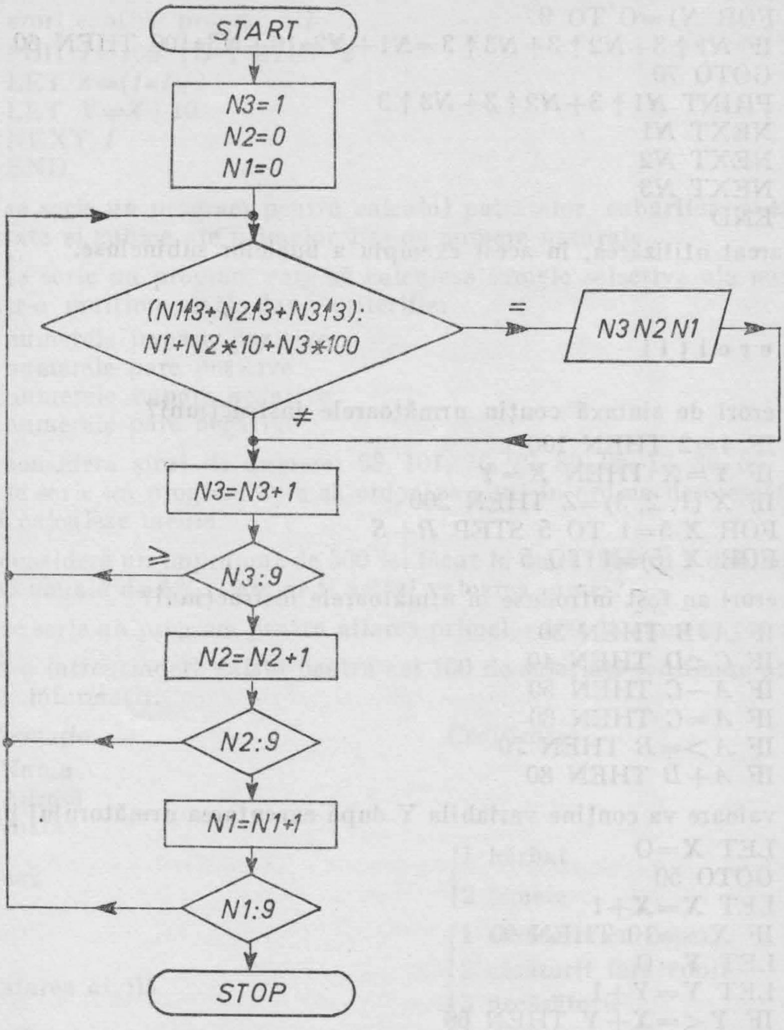


Fig. 7.3

Modul de rezolvare a problemei este indicat în diagrama logică din fig. 7.3. Se utilizează variabilele *N*1, *N*2, *N*3, care reprezintă cifrele numărului, respectiv unităţile, zecile, sutele.

Se testează pentru toate numerele de 3 cifre (100÷999) condiţia pusă de problemă:

$$n^3{}_1+n^3{}_2+n^3{}_3=n_3\times 100+n_2\times 10+n_1$$

Programul poate fi întocmit fie utilizînd instrucţiunea **IF**, fie instrucţiunile **FOR** şi **NEXT**. Soluţia aleasă este cu instrucţiunile **FOR** şi **NEXT**, fiind mai simplă.

```
10 FOR N3=1 TO 9
20 FOR N2=0 TO 9
```

```
30 FOR N1=0 TO 9
40 IF N1↑3+N2↑3+N3↑3=N1+N2*10+N3*100 THEN 60
50 GOTO 70
60 PRINT N1↑3+N2↑3+N3↑3
70 NEXT N1
80 NEXT N2
90 NEXT N3
100 END
```

de remarcat utilizarea, în acest exemplu a buclelor subincluse.

Exerciţii

1. Ce erori de sintaxă conţin următoarele instrucţiuni?

```
10 IF 4=2 THEN 100
15 IF Y=X THEN X=Y
20 IF X (1, 2, 3)=Z THEN 200
25 FOR X 5=1 TO 5 STEP R+S
30 FOR X (5)=1 TO 5
```

2. Ce erori au fost introduse în următoarele instrucţiuni?

```
10 IF A*B THEN 30
20 IF C<D THEN 40
30 IF A-C THEN 50
40 IF A=C THEN 60
50 IF A>=B THEN 70
60 IF A+B THEN 80
```

3. Ce valoare va conţine variabila Y după executarea următorului program:

```
10 LET X=0
20 GOTO 50
30 LET X=X+1
40 IF X>=10 THEN 90
50 LET Y=0
60 LET Y=Y+1
70 IF Y<=X+Y THEN 60
80 GOTO 30
90 END
```

4. Ce erori conţine următorul program:

```
10 LET I=1
20 IF I>10 THEN 100
30 LET J=1
40 IF I>5 THEN 80
50 LET K=I*J
60 LET J=J+1
70 GOTO 20
80 LET I=I+1
90 GOTO 40
100 STOP
110 END
```

5. Ce erori conține programul?

```
10 FOR I=100 TO 1 STEP 2
20 LET Z=(I*I)/2
30 LET Y=Z+10
40 NEXT I
50 END
```

6. Să se scrie un program pentru calculul patratelor, cuburilor, rădăcinilor pătrate și cubice ale primelor 100 de numere naturale.

7. Să se scrie un program care să calculeze sumele selective ale numerelor dintr-o mulțime dată după criteriile:
   — numerele impare pozitive
   — numerele pare pozitive
   — numerele impare negative
   — numerele pare negative

8. Se consideră șirul de numere: 98, 101, 76, 79, 89, 95, 17, 86, 85. Să se scrie un program care să ordoneze șirul în ordine descrescătoare și să-i calculeze media.

9. Se consideră un împrumut de 500 lei făcut în anul 1955 cu o dobîndă compusă anuală de 6%. Care ar fi astăzi valoarea sumei?

10. Să se scrie un program pentru aflarea primelor 100 de numere prime.

11. Într-o întreprindere există pentru cei 100 de salariați codificate următoarele informații:

Informația	*Codificarea*
a. Nume	
b. Adresă	
c. vîrstă	
d. sex	1 bărbat 2 femeie
e. Starea civilă	1 căsătorit cu copii 2 căsătorit fără copii 3 necăsătorit 4 divorțat
7. Vechime în muncă	1 pînă la 5 ani 2 pînă la 10 ani 3 pînă la 20 ani 4 peste 20 ani

Se cer următoarele statistici:
— numărul femeilor și bărbaților în fiecare stare civilă;
— numărul salariaților cu copii;
— numărul salariaților cu vechime mai mare de 20 ani;
— numărul salariaților cu vîrsta între 19—25 ani.

12. Considerînd că pe numele unui copil se depun la CEC în fiecare lună 100 lei cu dobîndă anuală de 3,5%, ce sumă va avea copilul la împlinirea vîrstei de 18 ani?

Capitolul VIII

FUNCŢII ŞI SUBRUTINE

8.1. INTRODUCERE

Funcţiile şi subrutinele constituie modaliăţi prin care programatorul poate utiliza seturi de instrucţiuni generalizate pentru rezolvarea anumitor probleme specifice [28].

Funcţiile, care reprezintă seturi de instrucţiuni apelabile prin anumite mnemonici, oferă posibilitatea realizării directe a anumitor operaţii, scutind programatorul de o muncă considerabilă. Există de asemenea, în afara setului de funcţii standard, posibilitatea ca programatorul să-şi definească o anumită funcţie pe care apoi s-o utilizeze ori de cîte ori este necesară în program.

Utilizarea subrutinelor permite realizarea unor programe modulare, programe uşor de depanat şi care prezintă o mare elasticitate [42].

8.2. FUNCŢII STANDARD

Funcţiile, sînt instrucţiuni create pentru realizarea unor operaţii specializate şi pot fi utilizate în orice expresie legală BASIC.

Formatul general al funcţiilor este:

numele funcţiei (expresie)

numele funcţiei — este un nume din trei litere care indică funcţia ce urmează a se executa; funcţiile standard ale limbajului BASIC sînt date în tabelul nr. 8.1.

expresia — reprezintă argumentul funcţiei şi poate fi o constantă, o variabilă sau o expresie aritmetică; unghiurile trebuie indicate în radiani.

Executarea unei funcţii se face astfel: întîi este evaluat argumentul şi apoi se execută asupra lui operaţia indicată de numele funcţiei.

Exemplul 1

10 LET X=SQR ($A \times B$)

se calculează produsul $a \cdot b$, se ridică la puterea 1/2 şi valoarea obţinută este atribuită variabilei x.

FUNCŢII ARITMETICE STANDARD

Nume	Funcţia realizată	
SIN (*X*)	sin *x*	
COS (*X*)	cos *x*	
TAN (*X*)	tg *x*	
ATAN (*X*)	arctg *x* $(-\pi/2 \leq$ ATAN $(X) \leq \pi/2)$	
LOG (*X*)	ln *x*	$(x \triangleleft 0)$
EXP (*X*)	e^x	$(-178 \leq x \leq 175)$
SQR (*X*)	$x^{1/2}$	$(x \geq 0)$
ABS (*X*)	$\|x\|$	

FUNCŢII DE SISTEM

Nume	Funcţia realizată
INT (*X*)	Partea întreagă a numărului *x*
RND (*X*)	Un număr oarecare între 0 şi 1
SGN (*X*)	Semnul lui *x* $\begin{cases} 1 \text{ p: } x>0 \\ 0 \text{ pt } x=0 \\ -1 \text{ pt } x \triangleleft 0 \end{cases}$
LEN (*X*)	Lungimea şirului de caractere *S*
DET (*X*)	Determinantul ultimei matrice inversate
SYS (*X*)	*x* fiind o cifră 0÷8 sistemul furnizează următoarele informaţii: 0 — ora zilei 1 — luna din an (1÷12) 2 — ziua din lună (1÷31) 3 — anul 4 — numărul terminalului 5 — timpul de utilizare al U.C. 6 — numărul de chemări ale sistemului I/0 7 — codul ultimei erori 8 — numărul ultimului fişier utilizat
TAB (*X*)	Tabulează la poziţia caracterului *x*
EOF (*X*)	Detectează sfîrşitul fişierului *X*

Exemplul 2

10 LET *K*=*A* (SQR (*L*+1)/10)
instrucţiunea 10, atribuie variabilei *k* valoarea expresiei

$$k=a\sqrt{\frac{l+1}{10}}$$

Exemplul 3

```
10 LET X=6
20 LET Y 1=EXP (X)
30 LET Y 2=LOG (X)
40 LET Y 3=SQR (X)
```

```
50 LET X=-6·3
60 LET Y 4=ABS(X)
70 LET Y 5=SGN (X)
80 PRINT Y 1, Y 2, Y 3, Y 4, Y 5
90 END
```

Programul din exemplul 3 realizează următoarele operații:

$$y_1=e^6;\ y_2=\ln 6;\ y_3=\sqrt{6};\ y_4=6\cdot 3;\ y_5=-1.$$

Valorile calculate pentru variabilele $y_1 \div y_5$ vor fi apoi tipărite la consolă

Exemplul 4

```
10 LET X=INT (38.26)
20 LET Y=SQR (X)
30 LET Y=SIN (X*Y)
40 LET Z=INT (SQR (INT (X/Y↑X)))
50 IF ATN (Z)>=TAN (X+Y) THEN 80
60 LET X=INT (Y)
70 GO TO 90
80 LET Y=INT (Y)
90 PRINT X, Y, Z
100 END
```

În urma executării acestui program, la consolă se vor tipări valorile variabilelor x, y, z calculate astfel:

$$x=38;\ y_1=\sqrt{38};\ y_2=\sin 38\cdot\sqrt{38},$$

$$z=\text{INT}\sqrt{\text{INT}\left(\frac{x}{y^x}\right)}$$

dacă arctg $z\geqslant \text{tg}\,(x+y_2)$ se va tipări pentru y valoarea y_2, în caz contrar se va tipări numai partea întreagă a lui y_2.

Exemplul 5

Rezolvarea triunghiului oarecare. Se consideră un triunghi ABC în care se cunosc:

$b=34.91$

$A=98.71$ grade $=\dfrac{\pi}{180}\,.\,98.71$ radiani

$B=49.97$ grade $=\dfrac{\pi}{180}\,.\,49.97$ radiani

$C=31.32$ grade $=\dfrac{\pi}{180}\,31.32$ radiani

Se caută lungimile laturilor a, c

Din teorema sinusurilor:

$$\frac{a}{\sin A}=\frac{b}{\sin B}=\frac{c}{\sin C}$$

rezultă:

$$a=b\frac{\sin A}{\sin B}; \qquad c=b\frac{\sin C}{\sin B}$$

Programul pentru calculul laturilor a şi c este următorul:

```
10 READ A, B, C, D, P
20 DATA 98.71, 49.97, 31.32, 34.91, 3.1416
30 LET A1=(D*SIN (A*(P/180) ) )/SIN(B*(P/180) )
40 LET C1=(D*SIN(C*(P/180) ) )/SIN(B*(P/180) )
50 PRINT A1, C1
60 END
```

8.3. DEFINIREA FUNCŢIILOR

Programatorul îşi poate defini propriile lui funcţii utilizînd instrucţiunea **DEF.**

Formatul instrucţiunii este:

n n n n DEF FNa(d)=expresie

DEF — reprezintă mnemonica care indică tipul instrucţiunii

FNa — reprezintă numele funcţiei; primele două litere sînt obligatorii (FN), iar a poate fi orice literă $A \div Z$.

d — reprezintă argumentul funcţiei şi poate fi orice variabilă aritmetică

expresie — poate fi orice expresie aritmetică legală, incluzînd chiar alte funcţii anterior definite; sînt admise pînă la 4 funcţii subincluse.

Definirea funcţiei este limitată la expresii care pot fi scrise pe cel mult o linie. Pentru expresii mai lungi trebuie utilizate subrutine.

Exemplul 1

```
10 READ  I, J, K, X, Y,
20 DEF   FNA(I)=INT(I)
30 DEF   FN (B(J)=J↑3+K
40 DEF   FN C(X, Y)=X↑2+2*X*Y+Y↑2
50 DEF   FN E(I, J, X, Y)=(FNA(I)+FNB(Y) )/FNC(X, Y)
60 DEF   FND=Y
70 DATA  5, 5, 5, 5, 5,
80 END
```

Au fost definite următoarele funcţii:

$A(i)=\mathrm{INT}(i)$

$B(j)=j^3+k$

$C(x, y)=x^2+2\,xy+y^2$

$E(i, j, x, y)=\frac{A(i)+B(j)}{C(x, y)}$

Exemplul 2

```
10 LET  P=3.14.16
20 DEF  FNR(X)=X*P/180
30 DEF  FNS(X)=SIN (FNR(X) )
40 DEF  FNC(X)=COS(FNR(X) )
50 FOR  X=0 TO 45 STEP 5
60 PRINT X, FNS(X), FNC(X)
70 NEXT X
80 END
```

Programul din exemplul 2 va tipări valorile calculate pentru sin x şi cos x, pentru arcele 0, 5, 10, 15, ..., 45°; transformarea în radiani este realizată prin funcţia FNR.

8.4. SUBRUTINE

Subrutina este în general folosită în cazul în care o parte din program urmează a fi utilizată de mai multe ori, eliminînd astfel necesitatea repetării setului de instrucţiuni. Subrutina este citită în interiorul programului principal, limbajul BASIC avînd pentru controlul ei două instrucţiuni **GOSUB** şi **RETURN**.

Instrucţiunea GOSUB determină un salt în program, la prima instrucţiune a subrutinei, iar instrucţiunea RETURN provoacă revenirea în programul principal la prima instrucţiune după GOSUB.

Formatul acestor instrucţiuni este următorul:

n n n n GOSUB etichetă

⋮

n n n RETURN

GOSUB — reprezintă mnemonica care indică un salt în program, la o subrutină;

etichetă — indică prima instrucţiune a subrutinei;

RETURN — reprezintă mnemonica care determină revenirea în programul principal; o subrutină poate conţine mai multe instrucţiuni RETURN, provocînd astfel revenirea din mai multe puncte.

Un program poate conţine mai multe subrutine subincluse cu condiţia ca acestea să fie complet subincluse.

Configuraţia de subrutine acceptate este prezentată în fig. 8.1.

Modul de utilizare al subrutinelor subincluse este ilustrat în exemplul 1.

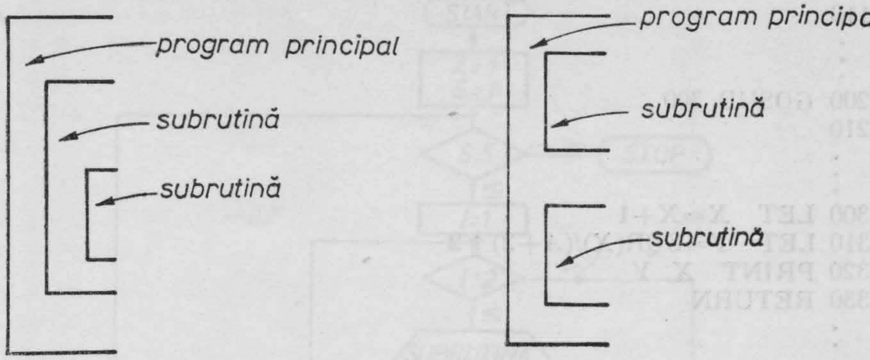


Fig. 8.1

Exemplul 1

```
10 LET X=A (Y)+M (N, I)
   .
   .
100 GOSUB 200
110 .
    .
200 LET  A(Y)=M(N, 1)
210 GOSUB 300
220 .
    .
240 RETURN
    .
    .
    .
300 PRINT  A(Y)
310 RETURN
1000 END
```

Se execută instrucţiunile 10 ÷ 99 ale programului principal, după care controlul este transferat subrutinei 200; după executarea primei instrucţiuni a acestuia are loc un salt la eticheta 300; detectarea instrucţiunii RETURN provoacă revenirea la 220; se continuă subrutina 200 şi se revine la instrucţiunea 110 a programului principal cînd este citită comanda RETURN de la eticheta 240.

O anumită subrutină poate fi apelată din diverse puncte ale programului principal; în exemplul 2 este ilustrată utilizarea de mai multe ori a unei subrutine:

Exemplul 2

```
10
 .
 .
100 GOSUB 300
```

```
110
 .
 .
200 GOSUB 300
210
 .
 .
300 LET  X=X+1
310 LET  Y=SQR(X)/(A+2)↑2
320 PRINT  X, Y
330 RETURN
 .
 .
1000 END
```

Execuţia instrucţiunilor 100, respectiv 200 provoacă transferul controlului la subrutina 300; instrucţiunile RETURN vor determina revenirea în programul principal la respectiv instrucţiunea 110, 210.

Exemplul 3

Simularea jocului cu zarul [35]

Programul din acest exemplu simulează următorul joc: doi jucători execută cîte două aruncări cu zarul; numerele obţinute la fiecare aruncare se sumează realizînd un punctaj; cîştigă cel cu punctajul mai mare. Simularea numărului obţinut la aruncare este realizată cu ajutorul funcţiei **RND**.

Programul este conceput pentru cinci jocuri.

Schema logică a programului este prezentată în fig. 8.2 şi 8.3.

Programul este următorul:

```
10 REM PROGRAM DE SIMULARE A JOCULUI CU ZAR
20 REM JOACA 2 JUCĂTORI: P şi G
30 REM CÎŞTIGA CEL CARE DIN DOUĂ ARUNCĂRI REALIZEAZĂ
       PUNCTAJ MAI MARE
40 REM SE POT JUCA 5 JOCURI
100 LET A=1
110 FOR S=1 TO 5
120 FOR I=1 TO 2
130 GOSUB 400
140 LET P=P+R
150 GOSUB 600
160 LET G=G+R
170 NEXT I
180 PRINT  P, G
190 IF P=G THEN 230
200 IF P>G THEN 250
210 PRINT "CISTIGĂ JUCĂTORUL G"
220 GO TO 250
```

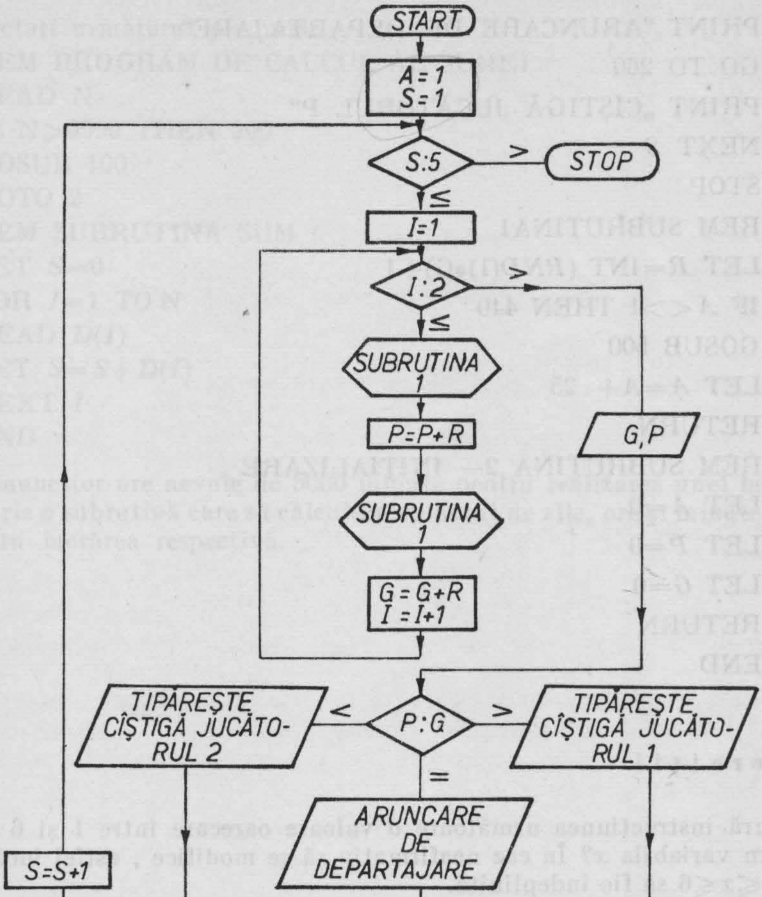

START
A=1
S=1
S:5
>
STOP
≤
I=1
I:2
≤
SUBRUTINA 1
P=P+R
G,P
SUBRUTINA 1
G=G+R
I=I+1
TIPĂREŞTE CÎŞTIGĂ JUCĂTORUL 2
<
P:G
>
TIPĂREŞTE CÎŞTIGĂ JUCĂTORUL 1
=
ARUNCARE DE DEPARTAJARE
S=S+1

Fig. 8.2

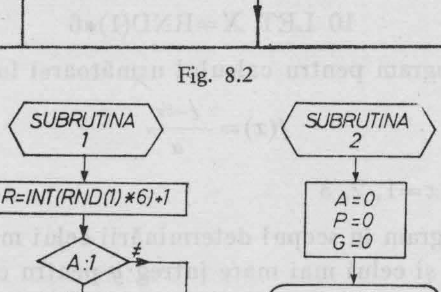

SUBRUTINA 1
R=INT(RND(1)*6)+1
A:1
≠
=
SUBRUTINA 2
A=A+.25
RETURN
SUBRUTINA 2
A=0
P=0
G=0
RETURN

Fig. 8.3

```
230 PRINT "ARUNCARE DE DEPARTAJARE"
240 GO TO 260
250 PRINT „CÎŞTIGĂ JUCĂTORUL P"
260 NEXT S
270 STOP
400 REM SUBRUTINA1
410 LET R=INT (RND(1)*G)+1
420 IF A<>1 THEN 440
430 GOSUB 500
440 LET A=A+.25
450 RETURN
560 REM SUBRUTINA 2– INIŢIALIZARE
510 LET A=0
520 LET P=0
530 LET G=0
540 RETURN
1000 END
```

Exerciţii

1. Asigură instrucţiunea următoare o valoare oarecare între 1 şi 6 inclusiv pentru variabila x? În caz neafirmativ să se modifice , astfel încît condiţia $1 \leqslant x \leqslant 6$ să fie îndeplinită.

   10 LET X=RND(1)*6

2. Să se scrie un program pentru calculul următoarei funcţii:

$$f(x)=\frac{e^{-bx}}{a}$$

   unde $a=5$, $b=2$, $x=1, 2, 3$

3. Să se scrie un program în scopul determinării celui mai mare întreg x pentru care $\sqrt{x}<100$ şi celui mai mare întreg y pentru care $e^y<100$

4. Corectaţi erorile de sintaxă, dacă acestea există, în următoarele instrucţiuni:

```
10 LET X=SQR(INT(RND(1)*52))
20 GOSUB 12345
30 LET RND(1)=SQR(Y)
40 GOSUB 7132
50 LET X=RND(X+Y)
```

5. Corectați următorul program:

```
  1 REM PROGRAM DE CALCUL AL SUMEI
  2 READ N
  3 IF N>9999 THEN 200
  4 GOSUB 100
  5 GOTO 2
100 REM SUBRUTINA SUM
110 LET S=0
120 FOR I=1 TO N
130 READ D(I)
140 LET S=S+D(I)
150 NEXT I
200 END
```

6. Un muncitor are nevoie de 5000 minute pentru realizarea unei lucrări. Să se scrie o subrutină care să calculeze numărul de zile, ore și minute necesare pentru lucrarea respectivă.

Capitolul IX

9. IMPRIMAREA REZULTATELOR

Facilităţile de extragere a datelor din calculator sînt concretizate în limbajul BASIC prin instrucţiunile **PRINT** şi **PRINTUSING.**

9.1. INSTRUCŢIUNEA PRINT

Instrucţiunea PRINT are formatul:

n n n n PRINT listă de expresii
n n n n ; listă de expresii

PRINT sau ; — reprezintă mnemonica care indică operaţia de tipărire a datelor;
lista de expresii — este o listă de variabile (simple sau indicînd masive de date), expresii aritmetice, sau şiruri de caractere.

Instrucţiunea permite tipărirea datelor fie în formă standard fie într-un format comandat de programator.

Formatul standard

O linie de tipărire este împărţită în 5 zone de cîte 15 spaţii; dacă elementele listei sînt separate prin virgulă atunci fiecare dintre acestea va fi tipărit începînd cu primul spaţiu al unei zone [65].

Exemplul 1

10 PRINT A, B, C
dacă $A=21.3$
$B=-7$
$C=.0003$
atunci la consolă se va tipări în următoarele poziţii:

```
0           14          28          42          56
↓           ↓           ↓           ↓           ↓
b21.3       -.7         b.0003
```

Formatul comandat

Dacă elementele listei sînt separate prin „;“ atunci acestea vor fi tipărite unul în continuarea celuilalt, lungimea zonelor fiind ignorată. Se obţine astfel o scriere mai compactă.

Lungimea zonei dorite poate fi fixată de programator la începutul lucrului la consolă utilizînd comanda TAB. (comanda TAB este descrisă în cap. 9.2).

Considerînd valorile din exemplul 1 și utilizînd instrucțiunea:

10 PRINT A; B; C;

se va tipări:

```
0             14            28
↓
ƀ21.3ƀ—.7ƀ.0003
```

Tipărirea numerelor

Orice număr sau constantă care poate fi reprezentat prin 6 cifre (respectiv 8 pentru dublă precizie) și punctul zecimal se tipărește fără utilizarea formei exponențiale; semnul se tipărește numai în cazul numerelor negative, în cazul numerelor pozitive lăsîndu-se un spațiu liber.

Orice număr care depășește 6 cifre este reprezentat în formă exponențială:

[—] n [.] n n n n n E+e [e]

n — reprezintă cifrele numărului
E — indică reprezentarea exponențială
e — cifră a exponentului
[] — indică parte opțională a reprezentării.

Dacă numărul este pozitiv, poziția semnului este lăsată liberă.

Exemple de tipărire a numerelor

Numărul	Tipărire simplă precizie	Tipărire dublă precizie
37514032	ƀ3.75140E+7	ƀ37514032
—.0001	—.0001	—.0001
173.275943	ƀ1.73275E+2	ƀ1.7327594E+3
—100100100	—1.00100E+8	—1.0010010E+8
.00071251	ƀ7.1251E-8	ƀ.00071251

Exemplul 2

```
10 LETX=2
20 PRINT X,(X*2)↑3,X↑4,
30 PRINT X*2, X—10, X—102
40 END
```

La consolă se va tipări:

```
0             14            28            42            56
↓             ↓             ↓             ↓             ↓
ƀ2            ƀ64           ƀƀ16          ƀ4            —8
—100
```

De remarcat că virgula de la sfîrșitul instrucțiunii 20 comandă amplasarea la început de zonă a primului element din instrucțiunea 30 PRINT.

Exemplul 3

```
10 LET X=625
20 PRINT X, „RĂDĂCINA PĂTRATĂ DIN X ESTE :" , SQR(X)
30 LET Y=SQR(X)
40 PRINT Y, „RĂDĂCINA PĂTRATĂ DIN Y ESTE :", SQR(Y)
50 END
```

Se va tipări:

```
0              14                      28            42
↓              ↓                       ↓             ↓
ƀ625           RĂDĂCINA PATRATĂ DIN X ESTE: ƀ25
ƀ25            RĂDĂCINA PATRATĂ DIN Y ESTE: ƀ5
```

În cazul în care variabila de tipărit Y (în ex. 3 șirul de caractere) depășește lungimea unei zone, elementul următor este tipărit la începutul primei zone libere.

Exemplul 4

```
10 LET X=2
20 PRINT X; (X*2)↑3; X↑4;
30 PRINT X*2; X-10; X-102
40 END
```

Se va tipări:

```
0    4    8    12   15   18
↓    ↓    ↓    ↓    ↓    ↓
ƀ2ƀƀ64ƀƀ16ƀƀ4ƀ-8ƀ-100
```

Comparînd cu ieșirea de la exemplul 2 se constată scrierea compactă determinată de utilizarea separatorului „;" în loc de „,".

Inexistența semnului de separare (, sau ;) la sfîrșitul unei instrucțiuni RPINT dintr-o secvență, determină pentru următoarea instrucțiune PRINT scrierea pe o linie nouă; se poate observa acest lucru pe exemplul următor:

Exemplul 5

```
10 LET X=2
20 LET P=3114.15926535
30 PRINT X,(b*2)↑3
40 PRINT P
50 PRINT X*2
60 PRINT X-10, X-102
70 END
```

Se va tipări

```
0              15
↓              ↓
ƀ2             b64
ƀ3.11415E+3
ƀ4
-8             -100
```

Exemplul 6

```
10 DIM E(10)
20 READ N$, G, N
30 LET S=0
40 FOR I=1 TO N
50 READ E(I)
60 LET S=S+E(I)
70 NEXT I
80 LET M=S/N
90 PRINT N$, G, N, M
100 DATA ION IONESCU, 234, 3
110 DATA 9, 8, 7
120 END
```

Programul realizează citirea numelui, grupei, numărului de examene susţinute şi notele obţinute pentru un student; se calculează media realizată şi apoi sînt tipărite aceste informaţii prin intermediul variabilelor *N*$, *G*, *N*, *M* respectiv.

La consolă se va tipări:

```
0              14           28            42
↓              ↓            ↓             ↓
ION IONESCU    ␢234         ␢3            ␢8
```

9.2. UTILIZAREA FUNCŢIEI TAB

Funcţia **TAB** are următorul format:

TAB (expresie)

expresia — este o expresie aritmetică evaluată ca un întreg.

Funcţia TAB (*X*) inserată în instrucţiunea PRINT va determina poziţionarea carului la coloana *x* şi tipărirea primului element începînd cu această coloană.

Dacă carul a depăşit poziţia respectivă, în momentul apariţiei funcţiei TAB, atunci aceasta este ignorată.

Exemplul 1

```
10 LET X=20
20 LET P=193
30 LET R=-.0000712
40 LET Q=-195.7362
50 PRINT P; TAB(X); R; TAB(30); Q
60 END
```

Se va tipări

```
0                  20          30
↓                  ↓           ↓
␢193              -7.12E-7    -1.95736E+2
```

Exemplul 2

Reluînd problema din exemplul 6 cap. 7.1, evidenţa rezultatelor studentului la examen, poate fi realizată într-o formă mai clară:

```
10 DIM E(10)
20 READ N$, G, N
30 LET S=0
40 FOR I=1 TO N
50 READ E(I)
60 LET S=S+E(I)
70 NEXT I
80 LET M=S/N
90 PRINT "NUMELE"; TAB(25); "GRUPA"; TAB(45); "MEDIA"
100 PRINT "STUDENTULUI"; TAB(45); "NOTELOR"
110 PRINT N$; TAB(25); G; TAB(45) M
120 DATA ION IONESCU, 234, 3
130 DATA 9, 8, 7
140 END
```

Rezultatele acestui program vor fi tipărite în următoarea formă:

```
0                         25                      45
↓                         ↓                       ↓
NUMELE                    GRUPA                   MEDIA
STUDENTULUI                                       NOTELOR
ION IONESCU               234                     8
```

Exemplul 3

Programul prezentat în acest exemplu calculează greutatea şi înălţimea medie pentru un grup de 12 studenţi; sînt tipărite, ca date de ieşire, greutatea şi înălţimea pentru fiecare dintre studenţi, precum şi mediile calculate.

```
10 REM PROGRAM DE CALCUL A ÎNĂLŢIMII ŞI GREUTĂŢII
20 REM MEDII PENTRU UN GRUP DE STUDENŢI
30 REM G(K) – GREUTATEA UNUI STUDENT
40 REM I(K) – ÎNĂLŢIMEA UNUI STUDENT
50 DIM G(100), I(100)
60 LET N=12
65 PRINT „K"; TAB(10); „GREUTATE"; TAB(20); „ÎNĂLŢIME"
70 FOR K=1 TO N
80 READ G(K), I(K)
90 PRINT K; TAB(10); G(K); TAB(20); I(K)
100 NEXT K
110 LET G1=0
115 LET I1=0
120 FOR K=1 TO N
130 LET G1=G1+G(K)
140 LET I1=I1+I(K)
150 NEXT K
160 LET M1=G1/N
190 LET M2=I1/N
200 PRINT "GREUTATEA MEDIE ESTE"; TAB(25); M1
210 PRINT "ÎNĂLŢIMEA MEDIE ESTE"; TAB(25); M2
```

```
220 DATA 72, 151, 69, 146, 68, 160
230 DATA 74, 176, 71, 185, 69, 157
240 DATA 69, 145, 70, 139, 74, 172
250 END
```

Rezultatele vor fi tipărite astfel:

```
0                10                20
↓                ↓                 ↓
K                GREUTATE          ÎNĂLŢIME
1                72                151
2                69                146
3                68                160
4                73                173
5                70                160
6                67                162
7                74                176
8                71                185
9                69                157
10               69                145
11               70                139
12               74                172
GREUTATEA MEDIE ESTE                  70,5
ÎNĂLŢIMEA MEDIE ESTE                  160.5
```

9.3. INSTRUCŢIUNEA PRINT USING

Instrucţiunea **PRINT USING** permite tipărirea variabilelor sau expresiilor conform cu un anume format specificat de programator.

Sintaxa instrucţiunii este următoarea:

n n n n PRINT USING şir de formate, listă de expresii

PRINTUSING — mnemonică care indică tipul instrucţiunii
şir de formate — poate fi un şir de literale anterior definite, sau un şir de variabile, specificînd formatele cîmpurilor în care urmează a se tipări valorile expresiilor din lista
lista de expresii — este o listă similară cu cea admisibilă în instrucţiunea PRINT (vezi cap. 7.1).

Reguli cu privire la format

1. Toate convenţiile de tipărire ale instrucţiunii PRINT sînt ignorate cu excepţia celei introduse de apariţia semnului separator (, sau ;) la sfîrşitul listei de expresii.

2. Se pot specifica unul sau mai multe cîmpuri de tipărire într-un şir de formate.

3. Specificarea unui cîmp se face utilizînd o combinaţie din următoarele caractere:

+ — #, . $

4. Caracterele de format pot apare chiar în interiorul unui şir de literale, compilatorul făcînd distincţie în funcţie de natura caracterului. Acest lucru este ilustrat prin următoarele exemple:

Exemplul 2

Reluînd problema din exemplul 6 cap. 7.1, evidenţa rezultatelor studentului la examen, poate fi realizată într-o formă mai clară:

```
 10 DIM E(10)
 20 READ N$, G, N
 30 LET S=0
 40 FOR I=1 TO N
 50 READ E(I)
 60 LET S=S+E(I)
 70 NEXT I
 80 LET M=S/N
 90 PRINT "NUMELE"; TAB(25); "GRUPA"; TAB(45); "MEDIA"
100 PRINT "STUDENTULUI"; TAB(45); "NOTELOR"
110 PRINT N$; TAB(25); G; TAB(45) M
120 DATA ION IONESCU, 234, 3
130 DATA 9, 8, 7
140 END
```

Rezultatele acestui program vor fi tipărite în următoarea formă:

```
0                       25                  45
↓                       ↓                   ↓
NUMELE                  GRUPA               MEDIA
STUDENTULUI                                 NOTELOR
ION IONESCU             234                 8
```

Exemplul 3

Programul prezentat în acest exemplu calculează greutatea şi înălţimea medie pentru un grup de 12 studenţi; sînt tipărite, ca date de ieşire, greutatea şi înălţimea pentru fiecare dintre studenţi, precum şi mediile calculate.

```
 10 REM PROGRAM DE CALCUL A ÎNĂLŢIMII ŞI GREUTĂŢII
 20 REM MEDII PENTRU UN GRUP DE STUDENŢI
 30 REM G(K) — GREUTATEA UNUI STUDENT
 40 REM I(K) — ÎNĂLŢIMEA UNUI STUDENT
 50 DIM G(100), I(100)
 60 LET N=12
 65 PRINT „K"; TAB(10); „GREUTATE"; TAB(20); „ÎNĂLŢIME"
 70 FOR K=1 TO N
 80 READ G(K), I(K)
 90 PRINT K; TAB(10); G(K); TAB(20); I(K)
100 NEXT K
110 LET G1=0
115 LET I1=0
120 FOR K=1 TO N
130 LET G1=G1+G(K)
140 LET I1=I1+I(K)
150 NEXT K
160 LET M1=G1/N
190 LET M2=I1/N
200 PRINT "GREUTATEA MEDIE ESTE"; TAB(25); M1
210 PRINT "ÎNĂLŢIMEA MEDIE ESTE"; TAB(25); M2
```

```
220 DATA 72, 151, 69, 146, 68, 160
230 DATA 74, 176, 71, 185, 69, 157
240 DATA 69, 145, 70, 139, 74, 172
250 END
```

Rezultatele vor fi tipărite astfel:

0	10	20
↓	↓	↓
K	GREUTATE	ÎNĂLŢIME
1	72	151
2	69	146
3	68	160
4	73	173
5	70	160
6	67	162
7	74	176
8	71	185
9	69	157
10	69	145
11	70	139
12	74	172

```
GREUTATEA MEDIE ESTE          70,5
ÎNĂLŢIMEA MEDIE ESTE         160.5
```

9.3. INSTRUCŢIUNEA PRINT USING

Instrucţiunea **PRINT USING** permite tipărirea variabilelor sau expresiilor conform cu un anume format specificat de programator.

Sintaxa instrucţiunii este următoarea:

n n n n PRINT USING şir de formate, listă de expresii

PRINTUSING — mnemonică care indică tipul instrucţiunii

şir de formate — poate fi un şir de literale anterior definite, sau un şir de variabile, specificînd formatele cîmpurilor în care urmează a se tipări valorile expresiilor din lista

lista de expresii — este o listă similară cu cea admisibilă în instrucţiunea PRINT (vezi cap. 7.1).

Reguli cu privire la format

1. Toate convenţiile de tipărire ale instrucţiunii PRINT sînt ignorate cu excepţia celei introduse de apariţia semnului separator (, sau ;) la sfîrşitul listei de expresii.

2. Se pot specifica unul sau mai multe cîmpuri de tipărire într-un şir de formate.

3. Specificarea unui cîmp se face utilizînd o combinaţie din următoarele caractere:

$$+ \; - \; \# \; , \; . \; \$$$

4. Caracterele de format pot apare chiar în interiorul unui şir de literale, compilatorul făcînd distincţie în funcţie de natura caracterului. Acest lucru este ilustrat prin următoarele exemple:

„VALOAREA ESTE $2.85"→$2.85 sînt caractere ale şirului
„VALOAREA ESTE $$$.##"→$$$.## reprezintă specificaţie de format
„REZULTATUL ESTE–643"→–643 sînt caractere ale şirului
„REZULTATUL ESTE –####"→–#### reprezintă specificaţia de format.

5. Formatul poate fi declarat printr-o variabilă şir, care apoi este utilizată în instrucţiunea PRINTUSING.

De exemplu:

```
10 DIM A$(10)
20 LET A$=„##.###"
30 PRINTUSING A$, 19.3, 7.5, .312
```

6. Terminarea formatului care descrie un cîmp este marcată prin apariţia primului caracter nespecific pentru format.

Exemplu:

„###bFOR b $$###.##"

(„###" → format; „$$###.##" → format)

7. Şirurile de caractere din lista instrucţiunii PRINTUSING pot fi de asemenea tipărite conform unui anume format în felul următor: pentru fiecare caracter al şirului se alocă un caracter în descrierea formatului; dacă şirul de caractere depăşeşte cîmpul descris de format, şirul este trunchiat.

Cele descrise aici sînt ilustrate în următoarea instrucţiune:

PRINTUSING „#.#.######", „NUMĂR CRT.", „STUDENT", „NOTA"

Se va tipări:

NUMĂRbCRT.STUDENTbbbNOTA

8. Formatele se utilizează în mod repetat, dacă şirul de formate conţine mai puţine elementele decît lista de expresii a instrucţiunii PRINT USING.

Se consideră următoarea instrucţiune:

PRINT USING „###b.....b#, #$", A, B, C, X, Y, Z, A1, B1

Elementele A, X, A1 vor fi tipărite conform formatului „###". Elementele B, Y, B1 vor fi tipărite conform formatului „.....". Elementele C, Z vor fi tipărite conform formatului „#, #$".

E x e m p l u

```
10 READ I, X(I), Y(I), N
20 LET A=X(I)+Y(I))/N
30 PRINT USING „#####", I, X(I), Y(I), A
40 DATA 2, 14.2, 11.8, 2
50 END
```

Se va tipări

2bbbb14.2b11.8b13

9. *Tipărirea numerelor*

a) *Format de tipul* ###

Fiecare caracter # reprezintă un spaţiu pentru o cifră.

Numărul este cadrat la dreapta

Semnul este ignorat

Se reprezintă numai numere întregi

În cazul în care numărul depăşeşte formatul se tipăresc*

Exemple:

Formatul	Numărul	Reprezentarea
#####	173	ƀƀ173
#####	−12	ƀƀƀ12
#####	11.34	ƀƀƀ11
#####	753912	*****

b) *Format de tipul* ##.###

Se pot tipări numere zecimale, poziţia virgulei zecimale fiind determinată de punctul din format.

Poziţiile părţii zecimale sînt întotdeauna completate (în caz de depăşirea formatului numărul este rotunjit, în caz contrar spaţiile libere sînt completate cu zerouri).

Depăşirea formatului la partea întreagă provoacă tipărirea*.

Exemple:

Formatul	Numărul	Reprezentarea
###.##	11	ƀ11.00
###.##	10.327	ƀ10.33
###.##	2934.25	******

c) *Format de tipul* ±##.###

Permite tipărirea numerelor cu semn.

Semnul poate fi tipărit la începutul sau la sfîrşitul numărului (caz în care se utilizează formatul ##.###±).

Efectul utilizării semnului + sau − este următorul:

+ determină tipărirea semnului + dacă numărul este pozitiv şi − dacă numărul este negativ;

− determină tipărirea semnului − dacă numărul este negativ şi lasă spaţiu liber dacă numărul este pozitiv.

Exemple

Formatul	Numărul	Reprezentarea
+##.##	12.98	+12.98
+##.##	7.05	+ƀ7.05
+##.##	−3.989	−ƀ3.99
+##.##	−175.34	******
##.##−	11.3	11.30ƀ
##.##−	−3.05	ƀ3.05−
##.##−	00.02	ƀ0.02ƀ

d) *Format de tipul* ++···
−−···

Permite alocarea mai multor poziţii pentru semn; acestea pot fi utilizate pentru cifrele numărului în caz de depăşirea formatului.

Exemple:

Formatul	Numărul	Reprezentarea
---·##	−15	−15.00
---·##	−100	******
---·##	8	ƀƀ8.00

e) *Formatul de tipul* ±$###.##

Semnul dolarului poate apare la primul sau al 2-lea caracter din lanţ şi determină tipărirea lui pe poziţia respectivă.

Exemple:

Formatul	Numărul	Reprezentarea
−$####.#####	−125.34	−$ƀ125.34000
−$####.#####	0.0291	ƀ$ƀƀ0.02910
$####.##−	15.399	$ƀƀ15.34−

f) *Format de tipul* ±$$ ...

Utilizarea de cel puţin două ori a semnului $, la începutul formatului, determină tipărirea lui în faţa primei cifre a numărului.

Exemple:

Formatul	Numărul	Reprezentarea
+$$$#.##	19.27	+ƀ$19.27
$$$##.##	−1.03	$0001.03−

g) *Format de tipul* ±##·##↑↑↑↑

Scrierea exponenţială este semnalată prin utilizarea a 4 săgeţi. Cîmpul exponenţial va fi tipărit prin E±n n, unde fiecare n reprezintă o cifră a exponentului.

Exemple:

Formatul	Numărul	Reprezentarea
+##·##↑↑↑↑	234.91	+23.49E+01
+##·##↑↑↑↑	−.02	−20.00E−03
++##.##↑↑↑↑	1000.68	100.07E+01

Exerciţii

1. **Introduceţi instrucţiunile necesare pentru tipărirea rezultatelor de la programele exerciţiilor propuse în capitolele precedente**
2. **Volumul segmentului de sferă este dat de relaţia:**

$$V=h^2\left(\frac{c^2+4h^2}{8h}-\frac{h}{3}\right)$$

unde h — înălţimea segmentului
c — lungimea corzii

Realizaţi un program pentru calculul mai multor segmente de sferă, asigurînd tipărirea într-o formă dorită a mărimilor h, c, v.

3. **Scrieţi un program care să citească pentru maxim 30 de studenţi numele şi notele obţinute la 5 examene, să calculeze media notelor şi media notelor pentru student. Se cere tipărirea rezultatelor sub forma unui tabel de tipul:**

Numele Disciplina 1 . . . Disciplina 5 Media

4. **Scrieţi un program pentru generarea calendarului anului 1976**
5. **Scrieţi un program pentru generarea calendarului pe următorii 5 ani.**

Capitolul X

INSTRUCŢIUNI MATRICIALE

Limbajul BASIC conţine un set de instrucţiuni care permit prelucrarea masivelor bidimensionale de date ca matrice. În anexa *B* sînt prezentate cîteva elemente cu privire la calculul matricial.

10.1. INSTRUCŢIUNI MATRICIALE DE INTRODUCERE A DATELOR ŞI DE ATRIBUIRE

Instrucţiunile matriciale pentru introducerea datelor sînt: MATREAD, MATINPUT, MATA=ZER, MATA=CON, MATA=IDN, MAT READ FILE, MAT INPUT FILE

10.1.1. INSTRUCŢIUNI DE CITIRE

n n n n MATREAD *A*, *B*,
n n n n MATREAD $A(d_1, d_2)$, $B(d_1, d_2)$

MATREAD — mnemonica care indică tipul instrucţiunii;
A, *B* — reprezintă numele matricelor ce urmează a fi citite; poate fi o listă de litere (*A*÷*Z*) separate prin virgulă;
$A(d_1, d_2)$ — reprezintă numele şi dimensiunea matricii; d_1 — numărul de linii; d_2 — numărul de coloane.

Exemplul 1

10 MATREAD *A*, *X*, *Z*

Executarea instrucţiunii are drept efect citirea unor blocuri de date bidimensionale în zonele de memorie rezervate pentru matricele *A*, *X*, *Z* respectiv.

Utilizarea acestei instrucţiuni presupune declararea anterioară a matricelor (de ex.: prin instrucţiunea DIM).

Citirea se face pe linii

```
10 DIM A(3,3), B(4,3)
20 MAT READ A, B
30 DATA 5, 11, 14, 12, 3, 9, 4, 12, 19
40 DATA 11, 4, 19, 12, 7, 99, 18, 17, 4, 63
50 END
```

Executarea acestui program va determina citirea următoarelor matrice:

$$A(3\times 3)=\begin{bmatrix} 5 & 11 & 14 \\ 12 & 3 & 9 \\ 4 & 12 & 19 \end{bmatrix} \qquad B(4\times 3)=\begin{bmatrix} 11 & 4 & 19 \\ 12 & 7 & 9 \\ 9 & 18 & 17 \\ 4 & 6 & 3 \end{bmatrix}$$

Citirea matricelor $A(3\times 3)$ şi $B(4\times 3)$ se poate realiza şi prin instrucţiunea MATREAD $A(3, 3)$, $B(4, 3)$ caz în care nu mai este necesară declararea matricelor prin instrucţiunea DIM.

Instrucţiunea MATINPUT are formatul

n n n n MATINPUT A, B, ...
n n n n MATINPUT A (d_1, d_2), **B**(d_1, d_2), ...

MATINPUT — reprezintă mnemonica care indică tipul instrucţiunii;
A, B, ... — lista numelor de matrice formate din litere ($A \div Z$) separate prin virgulă;
$A(d_1, d_2)$ — lista numelor de matrice dimensionate; d_1 — numărul de linii; d_2 — numărul de coloane.

Instrucţiunea permite introducerea la momentul execuţiei, de la tastatură a datelor pentru matricele respective.

Folosirea primei instrucţiuni presupune declararea anterioară a matricelor (de exemplu prin instrucţiunea DIM).

10.1.2. INSTRUCŢIUNI DE ATRIBUIRE

Instrucţiunile matriciale de atribuire sînt:

n n n n MATA = B
n n n n MATA = ZER
n n n n MATA = ZER (d_1, d_2)
n n n n MATA = CON
n n n n MATA = CON (d_1, d_2)
n n n n MATA = IDN
n n n n MATA = IDN (d_1, d_2)

MAT — reprezintă mnemonica care indică instrucţiunea matricială
A — nume de matrice (orice literă a alfabetului)
ZER — matricea 0
CON — matricea cu toate elementele 1
IDN — matricea unitate.

— Matricea A poate fi declarată anterior prin instrucţiunea DIM, sau chiar în cadrul instrucţiunii (d_1 reprezintă numărul de linii, d_2 numărul de coloane).
— În cazul primei instrucţiuni, matricele A şi B trebuie să aibe aceeaşi dimensiune.

Executarea acestor instrucţiuni provoacă respectiv: copierea matricii B în A, atribuirea matricii zero matricii A, introducerea în toate elementele matricei A a elementului 1, atribuirea matricei unitate matricei A.

Exemplul 1

```
10 DIM A(2, 3), B(2, 3)
20 MATREAD B
30 MAT A=B
40 DATA A, 0, 1, 2; 0, 3
50 END
```

În urma executării acestui program zona A a memoriei va conţine:

$$A=\begin{bmatrix}1 & 0 & 1\\ 2 & 0 & 3\end{bmatrix}$$

Exemplul 2

```
10 MAT A=ZER (3, 4)
20 MAT B=ION (4, )
30 MAT C=CON (2, 3)
40 MAT D=ION (3)
50 END
```

Structura matricelor A, B, C, D va fi respectiv:

$$A=\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\quad B=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad C=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}\quad D=\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}$$

10.2. INSTRUCŢIUNI ARITMETICE

Prin intermediul instrucţiunilor aritmetice se pot realiza direct următoarele operaţii pe matrice: adunarea şi scăderea, înmulţirea, înmulţirea cu un scalar, calculul inversei şi transpusei.

Instrucţiunile

n n n n MAT $A=B+C$
n n n n MAT $A=B-C$

realizează adunarea respectiv scăderea matricii C cu (din) B, dimensionarea matricei A în concordanţă cu matricea obţinută şi memorarea acesteia în A.

Matricele B şi C trebuie să aibe aceeaşi dimensiune. Într-o instrucţiune este permisă o singură operaţie; următoarea declaraţie este incorectă.

MAT $A=B+C-D$

Pentru realizarea celor două operaţii sînt necesare două instrucţiuni:

MAT $A=B+C$
MAT $A=A-D$

Instrucţiunea de multiplicare cu un scalar:

n n n n MAT A = **(expresie)**$*B$

expresia — poate fi orice expresie numerică corectă, inclusă în paranteze.

Elementele matricei B vor fi pe rînd multiplicate cu scalarul introdus prin expresia respectivă, iar rezultatul atribuit matricei A.

De exemplu

```
10 MAT A=(SQR(X))*B
10 MAT A=((U+V)/D)*B
```

Instrucţiunea:

n n n n MAT A=B*C

realizează înmulţirea matricelor B şi C şi atribuirea rezultatului matricii A; este necesară îndeplinirea condiţiei de dimensiune: $B(n\times m)$ şi $C(m\times p)$ adică numărul de coloane ale primei matrice să fie egal cu numărul de linii al celei de-a doua matrice. Matricea A va fi dimensionată $A(n\times p)$.

Matricea din membrul stîng al expresiei nu poate apare şi în membrul drept, deci o expresie de forma:

```
n n n n MAT A=A*B
n n n n MAT A=B*A
```

este incorectă.

Înmulţirea unei matrice cu ea însăşi impune condiţia ca matricea să fie pătrată:

```
n n n n MAT A=B*B
```

Matricea B este de dimensiune $n\times n$.

Exemplul 1

```
10 DIM A(3,3), B(3, 3), C(3, 3), D(3, 3), E(3, 3)
20 MAT READ A
30 MAT B=A
40 MAT C=A+B
50 MAT D=A-B
60 MAT E=A*B
70 DATA 1, 0, 1, -2, 2, 0, -1, 1, 0
65 MAT PRINT C, D, E
80 END
```

În urma execuţiei acestui program la consolă vor fi tipărite următoarele matrice:

$$C=A+A=\begin{bmatrix} 2 & 0 & 2 \\ -4 & 4 & 0 \\ -2 & 2 & 0 \end{bmatrix}, \quad D=A-A=\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$E=A\times A=\begin{bmatrix} 0 & 1 & 1 \\ -6 & 4 & -2 \\ -3 & 2 & -1 \end{bmatrix}$$

Instrucţiunea

n n n n MAT A **=TRN(**B**)**

realizează transpunerea matricei B (înlocuirea rîndurilor prin coloane) şi atribuie rezultatul lui A.

A şi B trebuie să fie zone distincte de memorie; deci o declaraţie de forma:

```
10 MAT A=TRN(A)
```

este incorectă.

Instrucţiunea

n n n n MAT A =INV(B)

determină calcularea inversei matricei B şi atribuie rezultatul matricei A. Matricea B trebuie să fie o matrice patrată.

Exemplul 2

```
10 DIM P(2, 2), Q(2, 2), R(2, 3), X(2, 3), Y(2, 2), Z(2, 2)
20 MATREAD P, Q, R
30 MAT X=P*R
40 MAT Y=INV(Q)
50 MAT Z=TRN(P)
60 MATPRINT X, Y, Z
70 DATA 4, 3, 2, 5, 1, 3, 2, 4
80 DATA 8, 7, 4, 3, 9, 6
90 END
```

Rularea programului va determina tipărirea la consolă a matricelor:

$$X=P*R=\begin{bmatrix}4 & 3\\2 & 5\end{bmatrix}*\begin{bmatrix}8 & 7 & 4\\3 & 9 & 6\end{bmatrix}=\begin{bmatrix}41 & 55 & 34\\31 & 59 & 38\end{bmatrix}$$

$$Y=Q^{-1}=\begin{bmatrix}-2 & 1.5\\1 & -.5\end{bmatrix}$$

$$Z=P^{T}=\begin{bmatrix}4 & 2\\3 & 5\end{bmatrix}$$

Exemplul 3

```
10 MAT C=ZER(2, 1)
20 MAT READ A(3, 1), B(3, 3), D(3, 3)
30 MAT C=B*A
40 MAT E=B-D
50 LET K=2
60 MATE=(K) E
70 MAT B=INV(B)
80 MAT Q(CON(3, 3)
90 MAT P=Q+B
100 MAT PRINT C, E, B, P
110 DATA 1, 2, 1, 1, 0, 1, 0, 2, 0, 0, 0, 1
120 DATA 1, 1, 1, -1, 0, 0, 0, 0, 3
130 END
```

Rezultatul calculelor executate prin acest program sînt matricele C, E, B, P care vor fi tipărite la consolă:

$$C=B*A=\begin{bmatrix}2\\4\\1\end{bmatrix} \quad E=K(B-D)=\begin{bmatrix}0 & -2 & 0\\2 & 4 & 0\\0 & 0 & -4\end{bmatrix}$$

$$B^{-1}=\begin{bmatrix}1 & 0 & 0\\0 & 1/2 & 0\\-1 & 0 & 1\end{bmatrix} \quad P=\begin{bmatrix}2 & 1 & 1\\1 & 3/2 & 1\\0 & 1 & 2\end{bmatrix}$$

Exerciţiul 4

```
10 DIM A(5, 5), B(5, 5), C(5, 5), D(5, 5) E(5, 5), F(5, 5)
15 DIM G(5, 5), H(5, 5)
20 READ I, J
30 MATREAD A (I, I), B(I, I), D(I, J), G(J, I)
40 MAT C=ZER(I, I)
50 PRINT "MATRICEA A DE ORDIN" I
60 MAT PRINT A;
70 PRINT "MATRICEA B DE ORDIN" I
80 MAT PRINT B
90 MAT C=A+B
100 PRINT "C=A+B"
110 MAT PRINT C;
120 MAT F=ZER (I, J)
130 MATF=C*D
140 MAT H=ZER(I, J)
150 MAT H=G*F
160 PRINT "MATRICEA H DE ORDIN" J
170 MATPRINT N
180 DATA 3, 1
190 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1, 2
195 DATA 3, 3, 2, 1
200 END
```

Este de remarcat faptul că prin instrucţiunea DIM au fost dimensionate toate matricile utilizate, iar în cursul programului acestea au fost redimensionate, acest lucru este posibil cu condiţia ca noile dimensiuni să fie inferioare celor declarate prin DIM.

Rezultatele apar tipărite la consolă astfel:

```
MATRICEA A DE ORDIN 3
                         1      2      3
                         4      5      6
                         7      8      9
MATRICEA B DE ORDIN 3
                         9      8      7
                         6      5      4
                         3      2      1
C=A+B
                        10     10     10
                        10     10     10
                        10     10     10
MATRICEA H DE ORDIN 1
                        360
```

Exerciţii

1. Corectaţi erorile introduse în următorul program:

```
10 DIM X(2, 2), Y(3, 3), A(3, 4)
20 MAT READ X, Y
```

```
30 MAT A=X+Y
40 MAT B=Y*A
50 DATA 0, 1, 2, 1, 0, 3, 1, 4, 0, 2
60 END
```

2. Corectați erorile introduse în următoarele instrucțiuni

```
10 MAT X=Y
20 MAT Y=Y+X-B
30 MAT X=X*Y
40 MAT B=TRN (X)
50 MAT A=INV(A)
60 MAT C=(K) C
70 MAT X=A-X
```

3. Scrieți un program care să calculeze determinantul unei matrice de dimensiune 3×5 multiplicate cu o matrice de dimensiune 5×3
4. Trei întreprinderi fabrică 4 produse obținînd beneficii diferite. În tabelul de mai jos sînt indicate cantitățile fabricate din fiecare produs pe fiecare întreprindere și beneficiile realizate pe produs. Scrieți un program pentru calculul beneficiului total obținut de fiecare întreprindere

Magazin	Producție				Beneficiu			
	I	II	III	IV	I	II	III	IV
1	5	3	4	9	0.5	0.7	0.1	0.2
2	6	2	7	6	0.4	0.6	0.1	0.3
3	4	4	8	4	0.45	0.5	0.15	0.25

Capitolul XI

ORGANIZAREA FIŞIERELOR DE INTRARE/IEŞIRE A DIRECTORILOR ŞI A PARTIŢIILOR PE DISC

Informaţia vehiculată în interiorul calculatorului, poate fi privită din două puncte de vedere: unul fizic, adică al purtătorului fizic de informaţie şi celălalt logic, adică al legăturilor dintre datele care constituie informaţia. Informaţiile dintr-un anumit volum (disc sau bandă magnetică), pot să nu fie corelate între ele putînd fi destinate pentru programe distincte şi utilizatori diferiţi. Informaţiile corelate între ele, formează o *înregistrare logică*. Un bloc de informaţie, (o înregistrare fizică) conţine una sau mai multe înregistrări logice. Totalitatea înregistrărilor logice cu aceeaşi destinaţie şi organizare, depuse pe un anumit suport, şi care pot fi tratate individual, formează un *fişier*, [61], [63], [65].

Acest termen se aplică la orice colecţie de informaţii sau la orice dispozitiv care primeşte sau care furnizează informaţii. Exemple: fişierul program sursă, fişierul binar relocabil, fişierul de listare, fişierul imagine de memorie (Save file), consola teletype, unitatea de casete magnetice.

Programul sursă este intrarea pentru asamblor, care produce la ieşire un fişier binar relocabil. Acesta constituie intrarea pentru încărcător (Loader), care încarcă şi relocă programul la locaţii absolute producînd un fişier de salvare (save file). Acest fişier este memorat pe disc cuvînt cu cuvînt, în ordinea în care va fi încărcat în memorie. În afară de încărcare (loading), mai există şi alte mijloace prin care un utilizator poate produce un fişier de salvare.

11.1. FIŞIERE PE BANDĂ ŞI PE DISC

Toate discurile magnetice şi fişierele pe disc, sînt accesibile prin numele de fişier. Un nume de fişier este un şir de caractere $\leqslant 10$ compus din următoarele caractere ASCII: caractere alfabetice, constante numerice şi $. Acest şir de caractere este terminat fie printr-o reîntoarcere a carului, fie printr-un spaţiu sau nul. Un nume de fişier poate conţine orice număr de caractere, dar sistemul va considera doar primele zece semnificative.

Un fişier va trebui să fie deschis şi anume asociat cu un canal RDOS, înainte ca să se poată avea acces la el. Un fişier pe disc, poate fi deschis complet, permiţînd mai multor utilizatori să aibă acces simultan şi să modifice conţinuturile fişierelor; fişierul poate fi deschis în mod exclusiv, permiţînd doar unui utilizator să modifice fişierul şi altor utilizatori să citească fişierul; un fişier poate fi deschis doar pentru citire pentru mai mulţi utilizatori.

11.2. EXTENSII DE NUME DE FIŞIER

Extensia care poate fi adăugată la un nume de fişier este un şir de caractere alfabetice şi trebuie să includă caracterul $. Extensia poate avea orice număr de caractere, dar sistemul le consideră doar pe primele două semnificative. Semnul (.) separă extensia de numele fişierului. Exemplu de nume de fişier cu extensie: FOO.P$. Componenta sistemului de operare CLI*, adaugă de multe ori extensii la numele de fişier, indicînd tipul informaţiei pe care acesta îl conţine şi distingîndu-l de alte tipuri de fişiere care rezultă de la acelaşi program sursă.

Astfel dacă un fişier sursă este numit A.SR, el poate produce următoarele tipuri de fişiere: [63], [65]

A.RB fişier binar relocabil (*r*elocatable *b*inary file)
A.SV imagine memorie (*sav*e file)
A.LS fişier de listare (*lis*ting file)
A.OL fişier de intreţesere (*o*ver*l*ay file)

Un utilizator nu va putea să-şi numească un fişier sursă de exemplu A.SV din cauza confuziei care s-ar produce cu fişierul de salvare care are extensia SV.

Dispozitivele de intrare/ieşire au nume de fişier rezervate special care încep foarte adesea cu caracterul $. În continuare, sînt listate numele dispozitivelor acceptate de RDOS.

$CDR cititor de cartelă perforată
CT*n* unitatea de casetă *n* Data General (*n*=0−7), prima unitate de control
DKO disc cu capete fixe NOVADISC, prima unitate de control
DP*n* unitatea de discuri *n* cu capete mobile
$LPT imprimată cu 80 sau 132 coloane
MT*n* unitatea *n* de bandă magnetică cu 7 sau 9 piste, prima unitate de control
$PTP perforator de bandă de hîrtie de mare viteză
$PTR cititor de bandă de hîrtie de mare viteză
QTY multiplexor asincron pentru comunicaţii de date
$TTI teletype sau consola terminalului cu display
$TTO imprimanta teletype-ului sau ecranul display-ului
$TTP perforatorul de la teletype
$TTR cititorul de la teletype

Utilizatorul poate să aloce numele începînd cu $ şi la alte fişiere decît tipul de dispozitive. Astfel comanda **XFER** este utilizată pentru a transfera conţinutul unui fişier la alt fişier. Există două argumente:

XFER *fişier sursă* *fişier destinaţie*

De exemplu: XFER $PTR A↵

Aici, conţinutul de pe banda de hîrtie montată pe cititorul de bandă este transferat la un fişier numit A.

Dacă: XFER P $PTR ↵
conţinutul fişierului P este perforat pe banda de hîrtie.

---

* CLI (Command Line Interpreter) este un program de sistem fie în partiţia background sau în foreground, care acceptă linii de comandă de la o consolă de sistem şi translatează intrările în comenzi pentru RDOS, [63].

11.3. ATRIBUTE ŞI CARACTERISTICI DE FIŞIER

Atributele de fişier sînt trăsături distinctive ale fişierelor care pot fi stabilite sau schimbate prin cereri către sistem, în cele mai multe cazuri prin intermediul lui CLI. Aceste atribute sînt:

P fişier permanent care nu poate fi şters sau redesemnat
S fişier de salvare (imagine memorie)
W fişier protejat la scriere, în care nu se poate scrie
R fişier protejat la citire, care nu poate fi citit
A fişier protejat la atribut. Atributele unui astfel de fişier nu pot fi schimbate. După ce atributul A a fost pus, el nu poate fi îndepărtat
N nu este permisă editarea de legături a fişierului
? Primul atribut care poate fi definit de utilizator (bitul 9)
& Al doilea atribut care poate fi definit de utilizator (bitul 10)

Caracteristicile de fişier sînt trăsături distincte ale fişierelor, care nu pot fi schimbate de către utilizator. Lista cu caracteristicile de fişier este:

D organizarea aleatorie a fişierului
C organizarea continuă a fişierului
I accesibil prin I/E directă (doar SYS.DR* şi MAP.DR** au acest atribut)
L intrare după editarea legăturilor. Această caracteristică este dată mai mult pentru intrările directorilor, decît pentru fişierele însăşi
T partiţie. Aceasta defineşte un fişier ca fiind o partiţie
Y director. Această caracteristică defineşte un fişier ca fiind un director

Utilizînd comanda LIST, este posibil de a obţine informaţii de la directorul unui fişier, despre unul sau mai multe fişiere.

Sistemul RDOS conţine un număr de fişiere de sistem protejate permanent, ca de exemplu $TTI. Utilizatorul trebuie să aibă grijă să nu supraîncarce fişierele sale cu atribute restrictive, mai mult decît este necesar. Astfel, un fişier cu atributul AP nu poate fi şters de către utilizator în nici un mod, cu excepţia reiniţializării sistemului.

11.4. FIŞIERE PE DISC

Sistemul de operare RDOS, acceptă unităţi de discuri cu capete fixe şi mobile. RDOS acceptă pînă la două unităţi de comandă pentru discurile cu capete fixe NOVADISC, fiecare cu pînă la 8 unităţi adresabile de 128K, 256K, 756K cuvinte memorie. Pot fi incluse de asemenea în sistem, pînă la 8 uintăţi de discuri cu capete mobile, cu 2 pînă la 20 suprafeţe de discuri pe unitate.

Fişierele pe discuri pot fi organizate în modurile secvenţial, aleatoriu sau contiguu.

Fişierele organizate secvenţial conţin blocuri seriale de 255 cuvinte fiecare, iar fişierele organizate aleatoriu şi contiguu, conţin blocuri de 256 cuvinte fiecare, aceste fişiere putînd avea o lungime de la 0 la maximum 33.423.360 bytes, respectiv 33.554.432 bytes.

---

* SYS. DR – numele unui director de fişier.
** MAP.DR – ghid director.

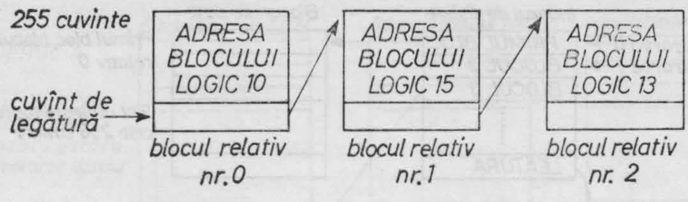


Fig. 11.1

legătura n este adresa blocului logic $(n-1)$. XOR. adresa blocului logic $(n+1)$.

11.5. FIŞIERE ORGANIZATE SECVENŢIAL

Aceste fişiere rezervă ultimul cuvînt de la fiecare bloc de 256 bytes pentru a fi folosit ca un punctator către următorul bloc de 256 bytes. Punctatorul este prevăzut doar pentru uzul sistemului, fiind transparent pentru utilizator. Fiecare bloc de 256 cuvinte, are o adresă unică, numită adresa blocului logic care este atribută lui de către sistem. Sistemul face deosebirea între adresa blocului logic şi adresele sectoarelor sau pistelor fizice. Mai există în mod distinct faţă de adresa blocului, numărul de bloc relativ, care indică poziţia relativă a unui bloc de date în interiorul unui fişier pe disc. Primul bloc în interiorul unui fişier pe disc este blocul relativ numărul zero, fig. 11.1.

11.6. FIŞIERE PE DISC ORGANIZATE SECVENŢIAL

În această formă de organizare, nu există adresele blocurilor logice. Blocurile memoriei pe disc, sînt secvenţiale în sensul că atunci cînd un bloc al unui fişier secvenţial a fost prelucrat, sistemul trebuie apoi să continue fie cu următorul bloc din sistem, fie cu blocul care a fost prelucrat imediat înaintea blocului prezent.

Toate transferurile de I/E ale fişierelor organizate secvenţial sînt înregistrate în zone tampon în interiorul sistemului. Numai blocurile în întregime sînt transferate şi fiecare bloc transferat este mai întîi citit într-o zonă tampon. Zona tampon de sistem, este organizată în blocuri de $(414)_8$ cuvinte fiecare.

Cînd se face citirea în această zonă, blocul neumplut complet şi cel mai vechi, este primul care va fi suprascris de către sistem. Cînd toate zonele tampon au fost umplute, atunci prima care a fost umplută, va fi prima care va fi suprascrisă. Întrucît toate intrările/ieşirile sînt realizate printr-o zonă tampon intermediară, este necesar mai mult timp pentru transferul de I/E.

11.7. FIŞIERE ORGANIZATE ALEATORIU

În organizarea aleatorie a fişierelor, este creat un index de fişier al tuturor adreselor blocurilor logice. Fiecare intrare în indexul de fişier este un cuvînt care adresează un bloc de pe disc.

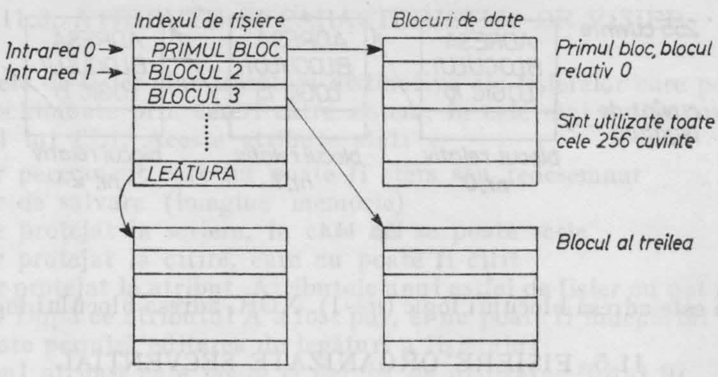


Fig. 11.2

Adresa maximă a blocului de pe disc, este $2^{10}-1$. Blocurile indexului de fişier, sînt legate în maniera unor blocuri secvenţiale. Blocurilor din fişierul organizat aleatoriu, le sînt atribuite numere de bloc relativ, de la 0 la *n*, unde fiecare număr este un întreg pozitiv într-o ordine secvenţială. În mod similar, fiecare intrare în indexul de fişier este considerată a fi la aceeaşi poziţie relativă în index, ca numărul relativ al blocului a cărui adresă logică o conţine. Astfel adresa logică a primului bloc într-un fişier aleatoriu, este găsită la intrarea numărul zero în indexul fişierului. Toate intrările zero în indexul de fişier, indică faptul că blocurile relative specificate, nu au fost scrise, fig. 11.2.

În prelucrarea fişierelor organizate aleatoriu, este nevoie de două accese la disc, pentru citirea şi scrierea fiecărui bloc: unul pentru indexul de fişier şi celălalt pentru blocul de date însuşi. Dacă indexul este rezident în memorie (fiind în prealabil citit într-o zonă tampon de sistem), este nevoie numai de un singur acces. Dacă blocul de date este rezistent, în memorie nu este necesar nici un acces la disc. Toate fişierele de salvare utilizează organizarea aleatorie. Toate comenzile de I/E care sînt accesibile pentru prelucrarea fişierelor organizate secvenţial, pot fi folosite pe fişierele organizate aleatoriu. Aceste comenzi vor cere în general, mai puţin timp pentru execuţia lor, datorită eficienţei dobîndite în folosirea organizării aleatorii a fişierului. Există un tip adiţional de comenzi de I/E folosite în prelucrarea fişierelor aleatorii: transferurile de I/E în bloc direct. În aceste transferuri, un întreg bloc de informaţie de pe disc, este transferat într-o zonă specificată de utilizator. Sistemul nu are nevoie de folosirea unor zone de sistem intermediare, mărindu-se astfel viteza de transfer. Totuşi, administrarea înregistrărilor în interiorul fiecărui bloc, devine o sarcină a utilizatorului.

11.8. FIŞIERE ORGANIZATE CONTIGUU

Acestea sînt fişiere la ale căror blocuri există un acces aleatoriu dar fără nevoia unui index de fişier aleatoriu. Fişierele contigue, sînt compuse dintr-un număr fix de blocuri pe disc, care sînt localizate la o serie neîntreruptă de adrese de bloc de disc. Aceste fişiere nu pot fi mărite niciodată ca dimensiune,

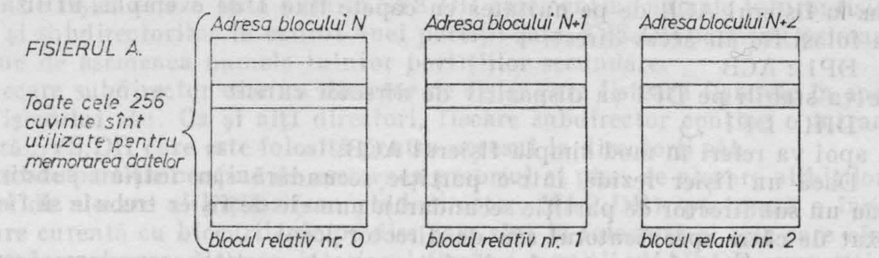


Fig. 11.3

nici micşorate. Întrucît blocurile de date sînt la adrese de bloc logice secvenţiale, tot ceea ce este necesar pentru a putea avea acces la un bloc în interiorul unui fişier contiguu este adresa primului bloc (sau numele fişierului) şi numărul blocului relativ în interiorul fişierului.

Toate operaţiile de I/E care pot fi realizate cu fişiere organizate aleatoriu, pot fi de asemenea realizate cu fişiere organizate contiguu, dar mărimea fişierului contiguu rămîne fixată. Fişierele organizate contiguu au avantajul că cer în mod uzual mai puţin timp pentru accesul la blocuri în interiorul fişierului, întrucît nu este nevoie de a citi un index de fişier.

11.9. REFERIREA FIŞIERELOR PE DISC

Fişierele pe disc sînt referite prin numere de fişier de pe disc. Întrucît un nume de fişier pe disc trebuie să rezide în unul din trei feluri diferite de directori* (ai partiţiei primare, ai partiţiei secundare sau al subdirectorului SYS.DR), sistemul trebuie de asemenea să fie prevăzut cu numele directorului care conţine numele de fişier dorit. Acest nume de director, poate fi livrat, fie prin prefixarea numelui de director la numele de fişier, fie prin stabilirea unui director curent prin intermediul lui CLIDIR, sau comenzii de sistem .DIR. Dacă există un director curent, toate celelalte referiri la nume de fişier vor fi direcţionate către acest director (în absenţa unui nume de director prefixat) pînă cînd un nou director este specificat.

Pentru realizarea unui fişier cu acces direct, un nume de fişier este prefixat de un specificator de director urmat de două puncte. Specificatorii de director pot fi globali sau locali:

Specificatorii de directori globali sînt mnemonice de unităţi de disc, care se referă la partiţiile primare de pe aceste dispozitive. Exemple de specificatori de directori globali:

DP<u>n</u> disc cu capete mobile, unităţile 0 pînă la 7

DK0 unitate de comandă pentru unitatea de disc cu capete fixe, unitatea logică 0

DK1 unitate de comandă pentru unitatea de disc cu capete fixe, unitatea logică 1

Astfel dacă ambele unităţi DK0 şi DP1 sînt în sistem, ambele vor fi iniţializate, DK0 fiind dispozitivul de director curent. Pentru a realiza accesul

---

* Noţiunea de director va fi explicată într-un paragraf următor.

său la fişierul ACB de pe unitatea cu capete fixe 1 de exemplu, utilizatorul va folosi fie un acces direct:

DP1 : ACB

fie va stabili pe DP1 ca dispozitiv de director curent

DIR DP1 ↵

şi apoi va referi în mod simplu fişierul ACB.

Dacă un fişier rezidă într-o partiţie secundară sau într-un subdirector (sau un subdirector de partiţie secundară), numele de fişier trebuie să fie prefixat de către specificatorul său de director local.

Specificatorii de directori locali sînt nume de partiţii secundare şi subdirectori. Fiecare specificator local, trebuie să fie urmat de două puncte. În absenţa unui nou specificator de director, toate celelalte referinţe la numele de fişier sînt îndrumate către directorul cel mai recent specificat în DIR sau în comanda .DIR.

Întrucît toţi directorii, cu excepţia directorilor partiţiilor primare sînt listaţi într-un director mai vechi (părinte), există o ierarhie printre specificatorii de directori [65]. Partiţiile primare sînt la nivelul cel mai ridicat. Urmează apoi subdirectorii de partiţii primare şi partiţiile secundare.

Nivel 0 Partiţia primară

↙ ↘

Nivel 1 Subdirectori partiţia secundară 1 partiţia secundară 2

↙ ↘ ↙ ↘

Nivel 2 Subdirector 1 subdirector 2 subdirector 3 subdirector 4

Dacă este necesar, partiţia primară însăşi trebuie să fie prima iniţializată. Astfel, dacă în schema de mai sus un utilizator doreşte să aibă acces la fişiere în subdirectorul 1 şi nici-un director din acest spaţiu de fişier nu a fost iniţializat, următorii directori trebuie să fie iniţializaţi în ordinea următoare: partiţia primară, partiţia secundară 1, subdirectorul 1.

11.10. PARTIŢIILE ŞI DIRECTORII DISCULUI

RDOS permite partiţionarea spaţiului fişierului pe disc între diferiţi utilizatori, după două baze: fixă şi semivariabilă. Partiţiile fixe ale unui fişier pe un spaţiu contiguu pe disc sînt denumite partiţii secundare. Partiţiile secundare sînt părţi exclusive reciproce ale spaţiului total al fişierului pe disc, partiţia primară. Spaţiul fişierului din interiorul unei partiţii poate fi alocat utilizatorilor pe o bază semi-variabilă. Aceasta înseamnă că utilizatorilor unei partiţii le pot fi alocate reciproc, exclusiv porţiuni din acest spaţiu al fişierului şi acele porţiuni pot fi extinse sau cuprinse ca mărime în cadrul limitelor spaţiului total al fişierului utilizabil în partiţia origine. Acele părţi variabile ale spaţiului fişierului sînt denumite subdirectori, fig. 11.4.

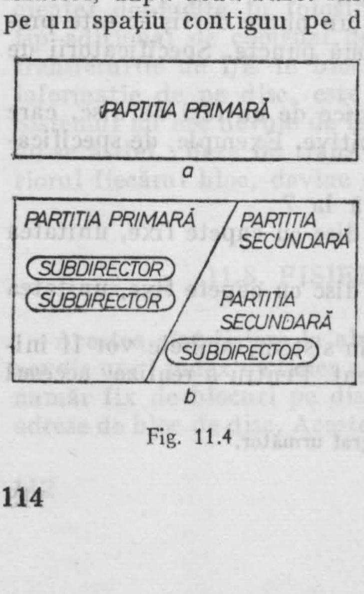


Fig. 11.4

Fiecare partiţie, primară şi secundară, conţine directorul său propriu de fişier. Un di-

rector de fişier este o listă numită SYS.DR care conţine numele tuturor fişierelor şi subdirectorilor în cadrul unei partiţii date. SYS.DR partiţiei primare conţine de asemenea numele tuturor partiţiilor secundare.

Fiecare subdirector este un director de fişier care listează fişierele în spaţiul fişierului său. Ca şi alţi directori, fiecare subdirector conţine o intrare numită SYS.DR care este folosită pentru accesul la directorii săi.

Fiecare partiţie conţine de asemenea propriul ei plan de alocare al biţilor. Planul de alocare al biţilor (sau ghid director, MAP.DR). păstrează o înregistrare curentă cu blocurile de pe disc care sînt în folosinţă şi cele care sînt libere să memoreze date, în interiorul fiecărei partiţii. MAP.DR a partiţiei primare reţine o înregistrare cu spaţiul total din disc, exceptînd blocurile de la 0 la 5 care pot conţine programul de încărcare automată. Astfel programul de încărcare automată a discului nu poate fi niciodată distrus, deoarece sistemul nu alocă acea porţiune a spaţiului din disc unde se află încărcarea automată. Acest lucru este valabil chiar dacă un anumit spaţiu din fişier poate fi alocat partiţiei secundare şi/sau subdirectorilor. Subdirectorii nu au un plan unic ca al directorilor dar utilizează MAP.DR al partiţiei origine.

Utilizatorii pot avea acces la un alt director în partiţia secundară şi în subdirectori prin intermediul intrărilor legate.

După iniţializarea completă a sistemului există numai o partiţie primară. Comanda CLI

CPART NAME *n*↵

este folosită pentru crearea partiţiei secundare, denumit NAME cu o lungime de *n* blocuri de disc contigue. Tot astfel, pentru a crea subdirectori, comanda CLI

CDIR SUBDIR↵

este utilizată în scopul creerii subdirectorului numit SUBDIR în partiţia curentă.

Iniţializarea parţială, de exemplu încărcarea automată a discului, nu distruge nici fişierele nici partiţiile secundare, sau subdirectorii.

Partiţia primară nu poate fi niciodată ştearsă. Subdirectorii şi partiţiile secundare pot fi totuşi oricînd şterşi. Dacă este ştearsă o partiţie secundară, subdirectorii conţinuţi de această partiţie sînt de asemenea şterşi.

Următorul exemplu compară partiţionarea discului cu utilizarea subdirectorilor, ca mijloc de împărţire a spaţiului din disc între un grup de 8 utilizatori. Poate fi folosită una din următoarele două scheme de alocare pentru a proteja fişierele fiecărui utilizator de a nu fi citite sau alterate de alt utilizator:

1. Crearea a 8 partiţii secundare şi atribuirea fiecărei partiţii unui utilizator

2. Crearea unei singure partiţii secundare (de 8 ori mai mare ca fiecare partiţie secundară de la punctul 1). Fiecărui utilizator îi este atribuit un subdirector distinct în cadrul partiţiei.

În ambele cazuri, 1) şi 2), fişierele pe disc ale fiecărui utilizator vor fi protejate şi fiecare utilizator va putea avea acces la fişierele din partiţia primară (asemănător software-ului utilitar). Procedeul 1 garantează un anumit spaţiu de fişier fix pentru fiecare utilizator. Dacă unul din utilizatori goleşte spaţiul său, el nu-şi poate însuşi spaţiul nefolosit al altei partiţii.

Procedeul 2 permite fiecărui utilizator să capete atît spaţiu de fişier, cît are el nevoie în interiorul partiţiei secundare comune, atîta timp cît există

spaţiu de fişier neutilizat. În cadrul acestui procedeu nici un utilizator nu poate fi asigurat de disponibilitatea unui spaţiu de fişier minim în orice moment, cu toate că spaţiul fişierului este folosit cu o eficienţă mai mare decît în procedeul 1.

11.10.1. ATRIBUIRILE INIŢIALE ALE BLOCURILOR DE DISC

Primele 20 de blocuri octale de pe disc, pe fiecare unitate de disc, au asignări* fixe, rămînînd cîteva blocuri libere fie pentru alte nevoi ale sistemului sau pentru memorarea fişierului utilizatorului. Blocurile 0—5 sînt rezervate pentru încărcarea automată. Blocul 6 este primul bloc index al fişierului aleator SYS.DR.

Blocurile 7—16 sînt rezervate pentru indexi de fişiere aleatoare în cazul în care apar schimbări în program în partiţia primară; blocul 17 este rezervat pentru primul bloc al fişierului continuu MAP.DR.

11.10.2. NUMĂRUL BLOCULUI DE DISC

0, 1, 2, 4, 5	Încărcare automată
6	Primul bloc de index pentru SYS.DR — fişier aleator
7, 8, 9, 10, 11, 12	Memorarea indexului de fişier modificat al programului din partiţia Background
13, 14, 15, 16	Memorarea indexului de fişier modificat al programului din partiţia Foreground
17	Primul bloc pentru MAP.DR.

Aşa cum s-a menţionat mai înainte, ghidul director, MAP.DR, este un fişier care indică ce blocuri din disc sînt în folosinţă curentă şi care sînt libere de alocat. Fiecare bit al fiecărui cuvînt din MAP.DR indică dacă un bloc specific este sau nu utilizat. Atribuirea blocurilor se face de la stînga la dreapta,

* Atribuiri.

în ordine crescătoare, începînd cu primul bloc al spaţiului contiguu de pe disc aflat în exploatare.

MAP.DR este un fişier contiguu a cărui mărime, în partiţia primară, este determinată de mărimea discului unde se află MAP.DR.

Cuvîntul	*Conţinutul*
0	Planul alocării blocurilor, 1 bit/bloc,
.	de la stînga la dreapta, în ordinea crescătoare a blocurilor,
.	începînd cu numărul de bloc 6
377	0=blocul este disponibil
	1=blocul este în folosinţă

11.10.3. DIRECTORII SISTEMULUI

Informaţia necesară în legătură cu fişierele dintr-o partiţie dată sau dintr-un subdirector este păstrată într-un fişier director de sistem, numit SYS.DR. Informaţia din fiecare SYS.DR include numele fişierelor, mărimea în bytes, a fişierelor şi însuşirile şi caracteristicile fişierelor.

Directorii de sistem folosesc un algoritm aleator pentru a accelera accesul la intrările directorului. De altfel, o arie iniţială de directori de sistem este alocată (la momentul în care sistemul este complet iniţializat) pentru intrările în partiţia primară pe un disc cu capete mobile. Această arie numită cadru este un set de blocuri contigue pe disc, setul fiind contiguu pentru a micşora timpul de acces al capului amovibil. Subdirectorii şi partiţiile secundare alocă memoria directorului de sistem după cerinţe.

Mărimea cadrului partiţiei primare este dependentă de tipul discului pe care este situată.

Următoarea listă de mărimea cadrului pentru diferite tipuri de discuri amovibile DGC.

Tipul unităţii	*Mărimea cadrului (în zecimal)*
4047	97
4048	193
4057	773

Structura SYR.DR-ului este identică pentru fişierul director de sistem şi pentru subdirectori.

Acest mod de organizare există pentru că SYS.DR este un fişier organizat aleator şi primul cuvînt al fiecărui bloc al fişierului reprezintă numărul fişierelor care sînt listate în acest bloc SYS.DR.

Acestui cuvînt îi urmează o serie de 228 de cuvinte-intrări, denumite descrierile fişierului utilizatorului sau UFD, care descriu fiecare fişier.

Urmează o exemplificare a cîtorva blocuri din SYS.DR:

Cuvînt	*Conţinut*
0	Numărul de fişiere în acest bloc al directorului (16_8 maximum)
1 . . 22	Descrierea fişierului utilizatorului

23 . . 44	Descrierea fişierului utilizatorului
. .	

Fiecare cuvînt de 22 octeţi pentru siguranţa intrărilor, are o structură fixă, arătată mai jos.

UFD-ul conţine informaţii despre fişierul care-i descrie numele, extensia de nume de 2 caractere însuşirile şi caracteristicile fişierului, mărimea fişierului, adresa primului bloc şi un cod al unităţii logice ce descrie unitatea asociată cu acest fişier.

Cuvînt	*Conţinut*
0—4	Numele fişierului
5	Extensia
6	Însuşirile
7	Atributele accesului de legătură
10	Contorul de bloc —1
11	Contorul de bytes în ultimul bloc
12	Prima adresă (de ex.: adresa logică a primului bloc din fişier)
13	Anul/ziua ultimului acces
14	Anul/ziua creerii
15	Ora/minutul creerii
16	UFD} informaţie variabilă
17	UFD}
20	Contorul utilizatorului
21	Legătura DCT*

Toate numele de fişier în cadrul fiecărui director de fişier trebuie să fie unice. O încercare de a introduce un nume de fişier la un director, unde mai există deja un acelaşi nume de fişier are ca efect apariţia unei indicaţii de eroare.

Un contor de fişier nenul indică că unul sau mai mulţi utilizatori au deschis fişierul.

Dacă apare o defecţiune de hardware, acest contor va fi adesea eronat şi trebuie făcut zero (via CLEAR) înainte ca fişierul său asociat să poată fi închis, redenumit sau şters.

11.10.4. STRUCTURA SUBDIRECTORULUI

Subdirectorii sînt subîmpărţiri comune a spaţiului unui fişier a partiţiei origine.

Deosebit de partiţia secundară, subdirectorii nu au definit un anumit spaţiu de fişier.

În schimb, subdirectorii îşi iau după necesităţi spaţiu de fişier de la partiţia origine şi eliberează spaţiul cînd nu mai este necesar.

---

* DCT = DEVICE CONTROL TABLE.

Subdirectorii creaţi recent conţin 3 blocuri: SYS.DR — blocul index iniţial, blocurile de date pentru SYS.DR şi intrările MAP.DR. Deoarece subdirectorii nu au spaţiul lor propriu de fişier, dar împrumută de la partiţia origine, intrarea ghidului director, în fiecare SYS.DR al subdirectorilor este o copie a intrării MAP.DR a partiţiei primare.

11.10.5. INTRĂRILE DE LEGĂTURĂ

RDOS oferă un mod elastic de acces, prin care utilizatorii pot avea acces la orice fişier pe discuri, pe bandă magnetică sau casete, prin denumirea acestuia sau prin nume diferite (denumite „alias"-uri).

În plus, utilizatorilor le este permis accesul la fişiere care nu fac parte din directorii proprii. Mecanismul care permite acest lucru este intrarea de legătură.

În cadrul unei aplicaţii uzuale intrările de legătură permit conservarea spaţiului din fişierul pe discuri, permiţînd ca numai o singură copie a unui fişier în uz comun să fie legată de către utilizatori, în acelaşi director, partiţie sau în alte partiţii.

Intrările de legătură pot indica alte intrări de legătură cu o precizie de rezolvare de maximum 10 cifre zecimale. Ultima intrare cu care se stabileşte legătura se numeşte intrarea de soluţionare. Intrările de legătură sînt create cu ajutorul comenzii CLI.LINK. sau prin comanda corespunzătoare de sistem LINK (fig. 11.5.)

Ori de cîte ori trebuie soluţionată o legătură (adică, atunci cînd legătura este deschisă), directorul care conţine intrarea de soluţionare este iniţializat de către sistem, dacă nu este deja iniţializat. Directorul care conţine intrarea de soluţionare nu trebuie să fie însă iniţializat în momentul creării unei intrări de legătură. Tot ce se cere pentru crearea unei intrări de legătură este ca denumirea ei să fie unică în directorul respectiv.

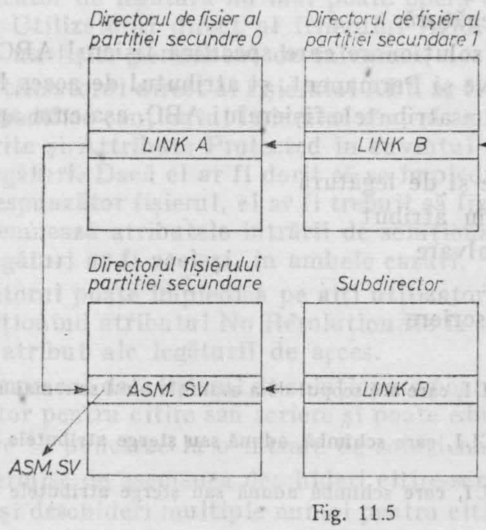


Fig. 11.5

În fig. 11.5, există patru legături la intrarea de soluţionare pentru ASM.SV*. Pentru ca o legătură dată să fie soluţionabilă, toate legăturile intermediare trebuie să fie de asemenea soluţionabile. Astfel, în ilustraţia precedentă, dacă LEGĂTURA B este anulată, LEGĂTURA C nu va fi rezolvată; LEGĂTURA A ar putea fi totuşi soluţionată în orice caz, deoarece nu există alte legături intermediare pînă la intrarea de soluţionare.

Fiecare intrare de soluţionare conţine două serii de atribute: atribute de intrări de soluţionare şi atribute de acces la legături. Atributele intrării de soluţionare se aplică utilizatorilor direcţi ai fişierelor. Atributele de acces la legături specifică atributele fişierului pentru utilizatori care stabilesc legătura cu aceste fişiere. Caracteristicile intrării de soluţionare (adică organizarea fişierului) se stabilesc cînd se crează fişierul de soluţionare, în cazul nostru ASM.SV; atributele intrării de soluţionare se poate să fi fost modificate între timp de către utilizator, prin CHATR-ul** comenzii de CLI (sau CHATR-ul comenzii de sistem). Cuvîntul care denotă atributele de acces la legături este de asemenea stabilit de către utilizator. Iniţial este 0, dar poate fi modificat prin CHLAT*** al comenzii CLI (sau .CHLAT. al comenzii de sistem).

Atributele de soluţionare generează accesul unui utilizator la un fişier deschis, în mod direct. Dar, atunci cînd utilizatorul deschide un fişier prin o intrare de legătură, utilizatorul consideră atributele fişierului de soluţionare ca fiind SAU-ul logic al atributelor fişierului de soluţionare şi al atributelor de acces la legături. Cu alte cuvinte, atributele unui fişier de soluţionare, văzute prin prisma unei intrări de legătură, sînt un compus format din atributele fişierului de soluţionare şi atributele de acces la legături.

După ce utilizatorul a deschis un fişier printr-o intrare de legătură, el poate modifica exemplarul pe care-l posedă din atributele fişierului, cu ajutorul unei comenzi CHATR. Se observă că această comandă nu permite ca atributele unui fişier protejat de acestea să fie schimbat. Deoarece fiecare utilizator de legături modifică numai exemplarul său din atributele fişierului, mulţi utilizatori de legături pot altera atributele respective. Cu toate acestea, orice alterare a acestor atribute este temporară şi rămîne valabilă numai pe durata deschiderii fişierului prin intrarea de legătură.

Fiind dată o intrare de soluţionare care specifică fişierul ABC ca avînd atributele de soluţionare Save şi Permanent, şi atributul de acces la legătură Write Protected, să examinăm atributele fişierului ABC, aşa cum apar acestor trei utilizatori, fig. 11.6.

UFD-urile de soluţionare şi de legătură

NOTA: 1B1 = Protejat prin atribut
1B2 = Fişier de salvare
1B14 = Permanent
1B15 = Protejat la scriere

---

* ASM este o comandă de CLI, care are scopul de a asambla unul sau mai multe fişiere sursă, folosind ansamblul relocabil.

** CHATR — Comandă de CLI, care schimbă, adună sau şterge atributele fişierului de soluţionare ale unui fişier dat.

*** CHLAT — Comandă de CLI, care schimbă adună sau şterge atributele de acces ale fişierului de legătură.

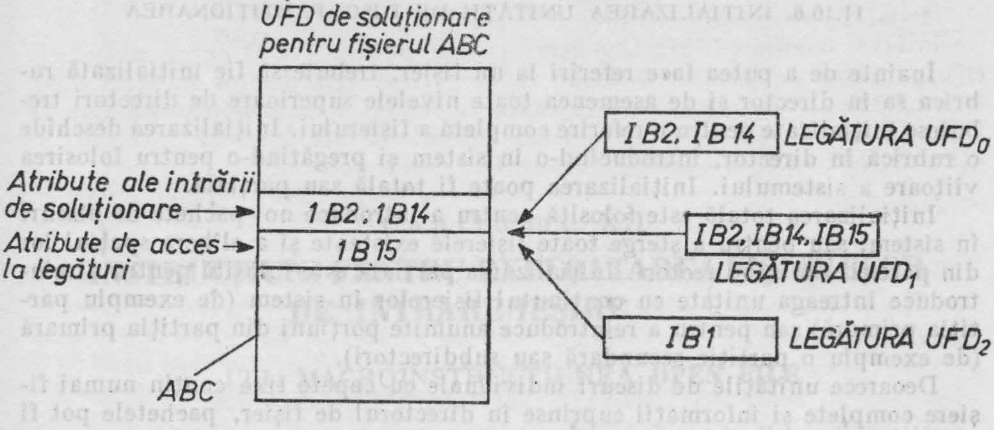


Fig. 11.6

Primul utilizator (cu UFD_0) a înlăturat atributul Write-Protected (protejat la scriere). Astfel, cuvîntul ce arată atributele, văzut de primul utilizator, conţine atributele Save şi Permanent. Se observă că, deoarece intrarea de rezolvare conţine atributul permanent, nici o legătură de utilizator nu poate anula acest fişier. Aceasta pentru că numai un fişier închis poate fi anulat, şi ori de cîte ori fişierul de soluţionare este închis, atributele sale iniţiale de soluţionare (inclusiv atributul permanent) sînt restabilite, prevenind astfel anularea fişierului.

Al doilea utilizator nu şi-a modificat copia sa de pe cuvîntul ce desemnează atributele; el vede atributele ca fiind SAU-ul atributelor fişierului de soluţionare şi al atributelor iniţiale de acces la legături: Save, Permanent şi Write Protected. Astfel, al doilea utilizator nu poate nici să anuleze nici să modifice conţinutul fişierului. Al treilea utilizator a modificat cuvîntul ce exprimă atributele, pentru a conţine un singur atribut: *Attribute Protected*. Astfel, acest utilizator de legătură nu mai poate opera alte modificări în atributele fişierului. Utilizatorul direct al fişierului ABC, continuă însă să considere fişierul ca un fişier permanent de salvare.

Dacă utilizatorul direct al fişierului ABC ar fi dorit să împiedice alţi utilizatori să modifice conţinutul fişierului său, el ar fi trebuit să stabilească atributele Write şi Attribute Protected în cuvîntul ce desemnează atributele de acces la legături. Dacă el ar fi dorit să se împiedice pe sine însuşi de a modifica necorespunzător fişierul, el ar fi trebuit să fixeze aceste atribute în cuvîntul ce desemnează atributele intrării de soluţionare; efectul asupra utilizatorilor de legături ar fi acelaşi, în ambele cazuri.

Utilizatorul poate împiedica pe alţi utilizatori să facă legătura cu fişierul ABC, poziţionînd atributul No Resolution fie la intrarea de soluţionare, fie la cuvintele atribut ale legăturii de acces.

La un moment dat, fişierul de soluţionare poate fi deschis de către cel puţin un utilizator pentru citire sau scriere şi poate exista cel puţin o intrare de legătură care să puncteze la o intrare de soluţionare.

Sînt permise de asemenea deschideri citire-scriere numai pentru un singur utilizator şi deschideri multiple numai pentru citire.

11.10.6. INIȚIALIZAREA UNITĂȚII DE DISC. PARTIȚIONAREA

Înainte de a putea face referiri la un fișier, trebuie să fie inițializată rubrica sa în director și de asemenea toate nivelele superioare de directori trebuiesc inițializate pentru o referire completă a fișierului. Inițializarea deschide o rubrică în director, introducînd-o în sistem și pregătind-o pentru folosirea viitoare a sistemului. Inițializarea poate fi totală sau parțială.

Inițializarea totală este folosită pentru a introduce noi pachete de discuri în sistem, sau pentru a șterge toate fișierele existente și a elibera spațiul lor din partiții sau subdirectori. Inițializarea parțială este folosită pentru a reintroduce întreaga unitate cu conținutul fișierelor în sistem (de exemplu partiția primară) sau pentru a reintroduce anumite porțiuni din partiția primară (de exemplu o partiție secundară sau subdirectori).

Deoarece unitățile de discuri individuale cu capete fixe conțin numai fișiere complete și informații cuprinse în directorul de fișier, pachetele pot fi extrase din sistem fără a afecta conținutul fișierului sau definițiile partiției și subdirectorului.

Deși mulți specificatori de directori pot fi inițializați, în orice moment nu poate să existe decît un singur „director lipsă“ curent.

Directorul lipsă curent este directorul în care sînt dirijate toate referințele la fișiere în absența unor informații suplimentare privind specificatorul din director.

Capitolul XII

INSTRUCŢIUNI PENTRU EXPLOATAREA FIŞIERELOR DE INTRARE/IEŞIRE

12.1. MACROINSTRUCŢIUNEA OPEN FILE

Scopul: Instrucţiunea **OPEN FILE** leagă un nume de fişier de utilizator sau un nume de dispozitiv periferic al sistemului, cu un număr de fişier de utilizator pentru referiri ulterioare de I/E. Această instrucţiune determină de asemenea modul în care este utilizat fişierul (citire, scriere, acces aleator, sau adăugarea de noi înregistrări) [42], [63], [65].

Formatul:

OPEN FILE [*num-expl*, *num-exp2*], *nume-fişier*

unde: *nume-fişier* este un literal, nume de fisier sau un şir de variabile evaluate ca nume de fişier

num-expl este o expresie numerică care dă numărul fişierului de utilizator, ce trebuie evaluat la un număr în gama 0–7 (deoarece 8 este limita canalelor de utilizator deschise simultan). Numărul fişierului este asociat cu numele de fişier şi este folosit pentru referinţe viitoare către fişier (pentru citire, scriere, închidere, etc.).

num-exp2 este o expresie numerică care dă modul în care fişierul este deschis, printr-un număr din gama 0–3. Fiecare mod este definit astfel:

Modul 0 – Acces aleator (I/E).

Numai fişierele pe disc trebuie să fie deschise în modul de acces aleator. În momentul cînd este deschis, un fişier cu acces aleator poate să fie citit sau scris. Dacă nici – un fişier avînd numele dat în instrucţiunea OPEN FILE nu este găsit în directorul de utilizator, va fi făcută în director o intrare pentru noul nume de fişier.

Modul I – Ieşire (scrie un nou fişier)

În acest mod poate fi deschis un fişier pe disc sau pe un dispozitiv corespunzător de ieşire. Sînt permise în acest caz, doar scrierile în fişier. Dacă un astfel de fişier cu acest nume există întotdeauna în directorul utilizatorului, copia prealabilă este mai întîi ştearsă de pe disc. În fiecare caz un nou fişier va fi creat în directorul utilizatorului iniţializat cu lungimea O.

Modul 2 – Ieşire (adăugare la un fişier scris deja)

Orice fişier de ieşire, poate fi deschis în vederea adăugării unei noi înregistrări (modul de lucru de adăugare), Cînd este deschis, fişierul este poziţionat către sfîrşitul fişierului curent, astfel încît datele suplimentare înscrise în fişier, îl vor extinde. Dacă fişierul nu există în directorul utilizatorului, va fi făcută o nouă intrare în directorul de utilizator pentru numele de fişier.

Modul 3 — Intrare.

În acest mod pot fi deschise atît un fişier pe disc cît şi pe un dispozitiv corespunzător de intrare. Dacă un fişier pe disc este deschis în acest mod, fişierul trebuie în prealabil să existe. În modul 3 pentru o deschidere de fişier sînt permise doar citirile. Dacă fişierul nu este găsit în directorul de utilizator, el va fi în continuare căutat în directorul public.

Exemple:

100 OPEN FILE [0,1], "TEST.1"
110 OPEN FILE [1,3], "$TR"
120 OPEN FILE [1,*M*], "S$"

12.2. MACROINSTRUCŢIUNEA CLOSE FILE

Scopul: Instrucţiunea CLOSE FILE anulează corespondenţa dintre un nume de fişier şi un număr de fişier de utilizator, astfel încît fişierul nu mai poate fi referit. Fişierele sînt închise cînd fişierul de I/E este complet. De asemenea poate fi necesar să se schimbe modul la un fişier deschis. Pentru a face aceasta, fişierul trebuie mai întîi închis şi apoi redeschis, folosind noul argument de mod. Format:

CLOSE FILE [*num-exp*$_1$]

num-exp$_1$ este numărul fişierului de utilizator asociat în prealabil cu un nume de fişier într-o instrucţiune OPEN FILE.

Exemple:

300 CLOSE FILE [2]
400 CLOSE FILE [I—5]

Macroinstrucţiunea CLOSE

Scopul: Instrucţiunea **CLOSE** va închide toate canalele deschise, în timp ce instrucţiunea CLOSE FILE va închide un canal specificat de utilizator. În condiţiile în care toate canalele sînt închise utilizarea instrucţiunii CLOSE nu provoacă apariţia mesajului de eroare Formatul:
CLOSE

Exemplu:

40 CLOSE

12.3. MACROINSTRUCŢIUNEA READ FILE

Scopul: Instrucţiunea **READ FILE** determină citirea datelor binare dintr-un fişier; acestea sînt atribuite variabilelor listate în instrucţiune [42], [61].

Formatul: Se prezintă în continuare două formate, unul folosit pentru citirea fişierelor sevenţiale, celălalt folosit pentru citirea unei înregistrări de pe un fişier aleator.

READ FILE [*num-exp*$_1$], *listă-variabile*
READ FILE [*num-exp*$_1$, *num-exp*$_2$], *listă-variabile*

unde:

listă-variabile este o listă de variabile numerice, ale căror valori trebuiesc citite de pe un fişier

num-exp$_1$ este o expresie numerică care evaluează pentru fişierul de utilizator, numărul unui fişier care fusese deschis în Modul 3 în cazul accesului secvenţial, sau în Modul 0 pentru accesul aleator.

num-exp$_2$ este o expresie numerică care evaluează numărul înregistrării care trebuie citită de pe un fişier cu acces aleator.

Comentarii: Fiecare variabilă din *listă-variabile* a instrucţiunii READ FILE, trebuie să corespundă tipului de date pentru fiecare valoare care este citită de pe fişier sau cu înregistrarea din interiorul fişierului. Dacă fişierul conţine atît valori numerice cît şi şiruri de caractere atunci variabilele de tip corespunzător, trebuie să fie date în ordine corectă în instrucţiunea READ FILE. În cazul citirii unui fişier cu acces aleatoriu, încercarea de citire a unei înregistrări care nu a fost niciodată scrisă, va cauza o înregistrare cu toate poziţiile zero.

Funcţia EOF poate fi folosită pentru a detecta sfîrşitul datelor, atunci cînd se transferă date de la un fişier în memorie. Funcţia EOF este utilizată în legătură cu o instrucţiune de transfer pentru a prevedea o instrucţiune la care să se facă transferul în cazul detectării sfîrşitului de fişier. Formatul acestei funcţii este:

EOF [*număr-fişier*]

număr-fişier este numărul unui fişier deschis pentru citire.

Funcţia EOF este evaluată la o valoare numerică întreagă 0 sau 1, indicînd dacă la ultima citire a fişierului, dat de *număr-fişier*, s-a detectat un sfîrşit de fişier. Dacă a fost detectat un sfîrşit de fişier, funcţia EOF ia valoarea „1", în caz contrar ia valoarea „0". Poate fi efectuat un transfer condiţionat, dacă funcţia EOF este folosită ca o expresie numerică într-o instrucţiune IF.

Exemple:

100 OPEN FILE [1, 3], "$*PTR*"
110 READ FILE [1], *A*, *B*, *C*, *D*, *E*, *F*, *G*
120 IF EOF (1) THEN 8000
190 OPEN FILE [2, 0], "*BB*"
200 READ FILE [2, 50], *X*, *Z*$, *Y*, *Z*

12.4. MACROINSTRUCŢIUNEA WRITE FILE

Scop: Instrucţiunea **WRITE FILE** determină extragerea datelor în format binar pe un fişier cu acces secvenţial sau o înregistrare a unui fişier cu acces aleator.

Formatul: În continuare vor fi prezentate două formate, din care primul permite scrierea datelor pe un fişier cu acces secvenţial, iar al doilea format este utilizat pentru scrierea datelor într-o înregistrare a unui fişier cu acces aleator.

WRITE FILE [*num-exp*$_1$], *listă-expresii*
WRITE FILE [*num-exp*$_1$, *num-exp*$_2$], *listă-expresii*

unde:

listă-expresii este o listă de expresii numerice sau un şir de variabile sau literali care evaluează valori numerice sau şiruri de valori pentru ieşire

num-exp$_1$ este o expresie numerică care evaluează numărul unui fişier de utilizator în prealabil deschis, în Modul 1 sau 2 pentru acces secvenţial, sau în Modul 0 pentru acces aleator

num-exp$_2$ este o expresie numerică care evaluează numărul înregistrării aleatorii care trebuie scrisă.

Comentarii: Şirul de variabile sau de literali utilizaţi de instrucţiunea WRITE FILE în modul secvenţial poate avea o lungime maximă de 132 baiţi. O înregistrare aleatorie, poate conţine maxim 128 baiţi. O expresie numerică are nevoie de 4 baiţi iar un şir de *n* baiţi, cere *n*+1 baiţi. Lungimea înregistrării este totalul tuturor baiţilor din *listă-expresii*.

Exemple:

```
0060 OPEN FILE [0, 1], "XX.2"
0070 FOR I=1 TO 50
0090 WRITE FILE [0], A[I], A[I] / I, S$, T$
  :
0500 OPEN FILE [1, 0], "DATA5"
0600 WRITE FILE [1,37], I, J, B[I] / A[I]
```

12.5. MACROINSTRUCŢIUNEA INPUT FILE

Scop: Această instrucţiune determină citirea datelor ASCII de pe un fişier în care datele sînt aranjate similar cu răspunsul unui teletype la o instrucţiune INPUT. Formatul:

INPUT FILE [*num-exp*$_1$], *listă-variabile*

unde:

listă-variabile este o listă de variabile numerice sau de variabile-şir*

num-exp$_1$ este o expresie numerică, ce evaluează pentru fişierul utilizatorului, numărul unui fişier în prealabil deschis în modul 3.

Exemple:

```
0040 FILE [1, 3], "$TPR"
0060 INPUT FILE [1], Z, Y, X, A$, B$
```

primele trei date care sînt citite de la cititorul de bandă de hîrtie trebuie să fie numerice şi ultimele două trebuie să fie şiruri de caractere.

Fişierul trebuie conceput cu separator între elemente; se utilizează fie virgula, fie comanda de întoarcere a carului.

Funcţia EOF, poate fi utilizată pentru a prevedea o instrucţiune la care să se facă transferul în cazul detecţiei sfîrşitului de fişier pe fişierul din care sînt introduse datele.

---

* trebuie asigurată concordanţa între tipul variabilelor listă şi al datelor asociate în fişier.

Exemplu:

```
0050 OPEN FILE [1, 3], "DATA"
0100 INPUT FILE [1], A, B, C, D, E, F, F1, F1$, F$, G [100]
0110 IF EOF (1) GOTO 1000
  .
  :
1000 PRINT "OUT OF DATA"
```

12.6. MACROINSTRUCŢIUNEA **PRINT FILE**

Scop: Această instrucţiune provoacă extragerea datelor în codul ASCII. Fişierul de ieşire produs este alcătuit, similar cu ceea ce se imprimă la terminal în urma executării instrucţiunii PRINT. Fişierul poate să livreze componentele sale, direct la un dispozitiv ce recunoaşte codul ASCII, ca de exemplu imprimate, sau pe un fişier pe disc, pentru o imprimare ulterioară off-line, [40], [42]. Formatul:

PRINT FILE [*num-exp*$_1$], *listă-expresii*

unde:

listă-expresii este o listă de expresii numerice, de variabile şi şiruri de literali, separaţi prin delimitatori speciali (, sau ; sau funcţia TAB).

num-exp$_1$ este o expresie numerică care evaluează pentru utilizator numărul unui fişier deschis în prealabil în modul 1 sau în modul 2.

Exemple:

```
0100 PRINT FILE [1], "OUT 6"
0360 PRINT FILE [0], "X="; "XSQR="; X↑2; "XCUBE=";
     X↑3
```

12.7. MACROINSTRUCŢIUNEA **PRINT FILE USING**

Scop: Instrucţiunea **PRINT FILE USING** determină ca valorile pentru expresiile date în instrucţiune să fie scoase pe un fişier în prealabil deschis, în formatul specificat de o expresie de tip şir dată în instrucţiune. Format:

PRINT FILE (*num-exp*$_1$], USING *expresie-şir*, *listă-expresii*

unde:

listă-expresii este o listă de expresii numerice, de variabile şi de şiruri de literali, ale căror valori trebuie să fie imprimate

expresie-şir, specifică formatul cîmpului în care valoarea fiecărei expresii trebuie să fie scoasă şi este identic cu specificaţiile *expresiei-şir* dată de instrucţiunea PRINT USING.

num-exp$_1$ este o expresie numerică care evaluează pentru fişierul utilizatorului, numărul unui fişier în prealabil deschis, fie în modul 1, fie în modul 2.

Exemplu:

```
0080 OPEN FILE [0,2], "T5"
0150 PRINT FILE [0], USING "++####.###", A, B, C,
     D, E, F
```

Ieşirea fişierului poate să fie concepută pentru o utilizare ulterioară la fel ca fişierul de intrare pentru instrucţiunea INPUT FILE, deoarece fiecare valoare este astfel formată, încît să se termine cu o virgulă.

12.8. MACROINSTRUCŢIUNEA **MAT READ FILE**

Scopul: Determină citirea datelor în format binar de pe un fişier, pentru zonele listate în instrucţiune. Aceste zone au putut fi în prealabil dimensionate, sau pot fi dimensionate în instrucţiunea **MAT READ FILE**.

Format: Primul dintre formatele prezentate mai jos este folosit pentru citirea fişierelor secvenţiale. Al doilea format este folosit pentru citirea unei singure înregistrări de pe un fişier cu acces aleator.

MAT READ FILE [*num-exp*$_1$], *listă-zone*

MAT READ FILE [*num-exp*$_1$, *num-exp*$_2$], *listă-zone*

unde:

listă-zone este o listă de zone ale căror valori vor fi citite din fişier

num-exp$_1$ este o expresie numerică care evaluează pentru fişierul de utilizator, numărul unui fişier care a fost deschis în modul 3 pentru accesul secvenţial sau în modul 0 pentru accesul aleator

num-exp$_2$ este o expresie numerică care evaluează numărul înregistrării ce trebuie citită de pe un fişier cu acces aleator.

Comentariu: Zonele care sînt în prealabil dimensionate trebuie să fie listate doar prin numele de zonă. Zonele care nu sînt deja dimensionate trebuie să fie dimensionate în instrucţiunea MAT READ FILE.

În citirea unui fişier cu acces aleator, o citire a unei înregistrări care nu a fost scrisă, va determina o înregistrare cu toate poziţiile nule.

Funcţia EOF, care a fost descrisă în legătură cu instrucţiunea READ FILE, poate fi folosită pentru a prevedea o instrucţiune la care să se facă transferul în cazul detecţiei „sfîrşitului-de-fişier", pe fişierul de la care au intrat datele. Argumentul funcţiei EOF este numărul fişierului deschis pentru citire

Exemplu:

```
0040 OPEN FILE [1, 3], "VALUES"
0060 MAT READ FILE [1], A, B, C(3, 4), D(5)
0090 IF EOF (1) GOTO 700
0700 Print "OUT OF VALUES"
```

12.8.1. MACROINSTRUCŢIUNEA **MAT WRITE FILE**

Scop: Instrucţiunea **MAT WRITE FILE** provoacă ieşirea datelor în conformitate cu zonele dimensionate în prealabil. Ieşirea se face în format binar la un fişier cu acces secvenţial, sau constituie o înregistrare a unui fişier cu acces aleatoriu.

Formatul: Primul format este folosit pentru a scrie într-un fişier cu acces secvenţial, iar al doilea format este folosit în scrierea unei singure înregistrări pentru un fişier cu acces aleatoriu.

MAT WRITE FILE [*num-exp*$_1$], *listă-zone*

MAT WRITE FILE [*num-exp*$_1$, *num-exp*$_2$], *listă-zone*

unde:

listă-zone în prealabil dimensionate, ale căror valori vor fi scrise pe fişier

num-exp$_1$ este o expresie numerică care evaluează pentru fişierul de utilizator, numărul unui fişier care fusese deschis în Modul 1 sau în Modul 2 pentru accesul secvenţial sau în Modul 0 pentru accesul aleatoriu.

num-exp$_2$ este o expresie numerică care evaluează numărul înregistrării aleatorii care trebuie scrisă.

Exemplu:

0050 OPEN FILE [0,1], "*AAA*"
0090 MAT WRITE FILE [0], *B*, *C*, *X*

12.9. MACROINSTRUCŢIUNEA **MAT INPUT FILE**

Scop: Această instrucţiune are ca efect citirea, pentru zonele listate în instrucţiune a datelor (în cod ASCII) dintr-un fişier. Zonele trebuie să fi fost în prealabil dimensionate sau vor trebui dimensionate în instrucţiunea MAT INPUT FILE. Formatul:

MAT INPUT FILE [*num-exp*$_1$], *listă-zone*

unde: *listă-zone* este o listă de zone ale căror valori trebuie să fie citite din fişier.

num-exp$_1$ este o expresie numerică care evaluează pentru fişierul de utilizator, numărul unui fişier care a fost deschis în prealabil în Modul 3.

Comentarii: Zonele care au fost în prealabil dimensionate trebuie să fie listate doar prin numele de zonă. Zonele care nu sînt deja dimensionate trebuie să fie dimensionate în instrucţiunea MAT INPUT FILE. Funcţia EOF, care a fost descrisă în legătură cu instrucţiunea READ FILE, poate fi folosită pentru a prevedea o instrucţiune la care să se facă transferul în cazul detecţiei sfîrşitului-de-fişier, pe fişierul de la care datele sînt introduse. Argumentul funcţiei EOF este numărul fişierului deschis pentru citire.

Exemplu:

0010 OPEN FILE [2, 3], "XX.*AA*"
0090 MAT INPUT FILE [2], (5, 5), *Y*, *Z*

12.10. MACROINSTRUCŢIUNEA **MAT PRINT FILE**

Scopul: Această instrucţiune determină ieşirea datelor conform zonelor dimensionate în prealabil. Ieşirea este în formatul ASCII pe un fişier cu acces secvenţial sau constituie o înregistrare a unui fişier cu acces aleator în cazul fişierelor pe disc, sau poate fi făcută pe un dispozitiv ce utilizează codul ASCII, ca de exemplu imprimanta. Formatul:

MAT PRINT FILE [*num-exp*$_1$], *listă-expresii*

unde:

listă-expresii este o listă de zone în prealabil dimensionate ale căror valori vor fi scrise în fişier.

num-exp$_1$ este o expresie numerică care evaluează pentru fişierul de utilizator numărul unui fişier care a fost deschis în Modul 1 sau 2 pentru accesul secvenţial sau în Modul 0 pentru accesul aleator.

Exemplu:

0010 OPEN FILE [0, 0], "*Z*.22"
0110 MAT PRINT FILE [0], *B*

12.11. INSTRUCŢIUNEA CHAIN

Scopul: Instrucţiunea **CHAIN** constituie un mijloc de chemare a unui program BASIC de pe disc sau de pe alt dispozitiv de intrare, din programul rulat curent. Utilizînd această instrucţiune, pot avea loc următoarele situaţii:

· Dacă programul este pe disc, sistemul caută directorul utilizatorului pentru *nume fişier*; dacă îl găseşte, sistemul va căuta directorul bibliotecii pe disc.

· Programul utilizatorului care este în mod curent rulat este scos din memorie dacă programul este găsit şi noul program este încărcat în memorie. Dacă nu este găsit *nume fişier*, programul curent rămîne în memorie.

· Noul program încărcat este rulat, de la instrucţiunea cu numărul cel mai mic din noul program. În mod opţional, utilizatorul poate să specifice dacă controlul va fi transferat la noul program, folosind CHAIN *nume fişier* THEN GO TO. Astfel utilizatorul poate să specifice unde va începe execuţia, în alt loc decît acela cu numărul cel mai mic de instrucţiune. Formatul:

CHAIN *nume fişier* **[THEN GO TO** *nr. instrucţiunii*]

unde: *nume fişier* este un literal, nume de fişier, sau un şir de variabile care evaluează numele fişierului.

nr-instrucţiune este orice număr de instrucţiune care există în programul specificat cu numele *nume fişier*.

Exemple:

```
0100 Chain "SUB1"
0230 CHAIN "Z$"
0190 CHAIN "SQET" THEN GO TO 0237
```

Programul înlănţuit, trebuie să fie în formatul fişierului SAVE.

12.12. INSTRUCŢIUNEA SAVE

Scopul: Această instrucţiune determină ca programul curent (instrucţiunile sursă şi datele) să fie scrise în format binar la un dispozitiv de ieşire binar, ca de exemplu perforator binar de bandă de hîrtie sau într-un fişier pe disc. Dacă scrierea se face pe disc, numele de fişier intră în directorul utilizatorului, înlocuind oricare fişier cu acelaşi nume. Un program „salvat" (SAVED) poate fi reîncărcat folosind instrucţiunea LOAD, instrucţiunea CHAIN, sau prin comanda RUN *nume fişier*. Salvarea unui program în format binar (se utilizează mai frecvent în formatul ASCII comanda LIST) este recomandată ca un mijloc de salvare al programului într-un format compact, prin aceasta reducîndu-se „overhead-ul" sistemului. În plus, un program care trebuia să fie executat parţial poate să fie salvat în acest format, astfel ca mai tîrziu prin executarea instrucţiunii LOAD să poată fi reluată execuţia sa. Deci, totul se întîmplă ca şi cînd programul nu ar fi fost vreodată îndepărtat din sistem. De asemenea un program salvat care este reîncărcat poate fi editat şi listat mai tîrziu în formatul ASCII. Format:

SAVE *nume-fişier*

unde:

nume-fişier este numele unui dispozitiv la care programul curent va trebui scris, sau numele sub care va fi înmagazinat în directorul de utilizator dacă programul curent trebuie scris pe un fişier pe disc.

Exemple:

0100 SAVE "*FA.BC*"
0233 SAVE "*$PTP*"
0555 SAVE "*CTO:2*"
0725 SAVE *S*$(1, 7)

12.13. INSTRUCŢIUNEA ENTER

Scopul: Folosirea ei provoacă intrarea instrucţiunilor BASIC conţinute în fişierul ASCII dat de *nume fişier* în programul curent.

Cînd o instrucţiune din fişier are acelaşi număr de ordine cu o linie din programul curent, linia „Intrată“ (ENTERED) va înlocui linia curentă. Acolo unde instrucţiunile în fişier au etichete diferite de acelea ale programului curent, instrucţiunile vor fi înserate în secvenţa lor proprie în programul curent. Utilizatorul poate să scrie sau să editeze linii în programul curent, folosind fişierul ASCII ca intrare, într-un mod asemănător ca în cazul în care el ar fi realizat intrarea pentru noile linii de program la teletype.

Fişierul pentru a putea fi introdus (ENTERED) trebuie să fi fost creat printr-o comandă de fişier LIST (a se vedea sintaxa comenzii LIST) sau creat la fel ca ieşirea unui alt program BASIC care foloseşte instrucţiunile PRINT FILE, sau care ar fi putut fi creat în afara sistemului Extended BASIC. Pentru a putea introduce un anumit program, poate fi utilizat orice dispozitiv de intrare în codul ASCII. Dacă nume fişier este un fişier pe disc, sistemul BASIC va căuta mai întîi directorul utilizatorului pentru numele de fişier. Dacă acesta nu este găsit acolo, se va căuta în directorul bibliotecii pe disc. Mesajele de eroare vor fi emise, fie dacă fişierul nu există, fie dacă nu este în format sursă — ASCII.

Porţiunea de date a unui program care se execută este neperturbată de către comanda ENTER, adică atribuirea variabilelor rămîne fixată. Astfel, instrucţiunea ENTER prevede o facilitate pentru rularea subprogramelor ca intreţeserile cu toate variabilele de program declarate ca „comune“. Format:

ENTER *nume fişier*

unde:

nume fişier este numele unui fişier pe disc sau al unui dispozitiv de intrare/ieşire care conţine instrucţiuni BASIC în formatul ASCII.

Exemple:

0055 ENTER "*$CDR*"
0010 ENTER "*LINES.BC*"

12.14. INSTRUCŢIUNI DE ÎNTREŢINERE A DIRECTORILOR

Cînd se utilizează compilatorul BASIC în legătură cu discul, se întreţin acei directori care conţin numele de fişier sau de fişiere, de pe disc împreună cu mărimea cuvintelor fiecărui fişier. Fiecare utilizator deţine propriul său director care conţine numele fişierelor sale. Există de asemenea un director de bibliotecă, ce conţine numele fişierelor care sînt accesibile tuturor utilizatorilor [40], [63], [65].

Instrucţiunile care sînt legate de directori, sînt acelea care şterg numele de fişier dintr-un director, şi care definesc un nou nume de fişier în director.

Există în acest sens, instrucţiunile DELETE şi RENAME, care vor fi tratate în continuare.

12.15. INSTRUCŢIUNEA **DELETE**

Scopul: De a şterge fişierul numit *nume fişier* din directorul de utilizator, ştergînd astfel efectiv fişierul de pe disc. Formatul:

DELETE *nume fişier*

unde:

nume fişier este numele fişierului din directorul de utilizator.

Exemple:

0100 DELETE "AO.1"

0518 DELETE "TEST"

12.16. INSTRUCŢIUNEA **RENAME**

Formatul :

RENAME *nume fişier 1*, *nume fişier 2*

unde:

nume fişier 1 este numele curent de fişier.

nume fişier 2 este noul nume care înlocuieşte *nume fişier 1* în directorul de utilizator.

Scopul: Această instrucţiune înlocuieşte un nume de fişier în directorul de utilizator, cu un nou nume.

Exemple:

0186 RENAME "TEST", "SQRT2"

0876 RENAME S$, "A"

12.17. COMENZI PENTRU FIŞIERE DE I/E

Utilizatorul poate emite comenzile direct de la terminal. Una din funcţiile modului de operare consolă, „keyboard", este specificarea fişierului de I/E şi întreţinerea directorilor. Comenzile de consolă încep cu un cuvînt de comandă care poate fi urmat de argumente, şi se termină cu o comandă de întoarcere a carului. Unele comenzi, sînt versiuni de consolă ale anumitor instrucţiuni BASIC. Compilatorul BASIC le recunoaşte drept comenzi, întrucît nu sînt precedate de etichetă.

12.18. COMENZI DE ÎNTREŢINERE A DIRECTORILOR

Cînd se foloseşte compilatorul BASIC în legătură cu discul, sînt menţinuţi un număr de directori şi anume aceia care conţin numele de fişier sau de fişiere pe disc, împreună cu mărimea în cuvinte a fiecărui fişier. Fiecare utilizator are propriul său director care conţine numele fişierelor sale, acesta din urmă numindu-se director de utilizator. În plus există un director de bibliotecă, care conţine numele acelor fişiere care sînt accesibile tuturor utilizatorilor. Comenzile de consolă, care sînt legate de directori, sînt acelea care listează numele de fişier conţinut fie în directorul utilizatorului fie în directorul bibliotecii. Ele se numesc FILES şi LIBRARY.

Comanda **FILES**

Scopul: Comanda determină imprimarea la terminal a listei tuturor numelor de fişiere din directorul de utilizator. Numele de fişier, atunci cînd sînt imprimate, vor fi separate de o etichetă. Formatul:

FILES

Exemplu:

FILES

Comanda **LIBRARY**

Scopul: Comandă imprimarea pe terminal a listei numelor de fişiere conţinute în directorul bibliotecii de pe disc. Numele de fişier, atunci cînd sînt imprimate, nu sînt separate de etichetă. Formatul:

LIBRARY

Exemplu:

LIBRARY

Comanda **LOAD**

Scopul: Această comandă execută un NEW implicit. Fişierul specificat este citit în memorie, devenind program curent. Fişierul numit poate fi pe disc sau poate fi pe un dispozitiv de intrare binar, ca de exemplu cititorul de bandă de hîrtie. În toate cazurile, doar un fişier care în prealabil a fost salvat va putea fi încărcat. Dacă se specifică un fişier pe disc, compilatorul BASIC cercetează mai întîi directorul utilizatorului. Dacă numele de fişier nu este în directorul utilizatorului, compilatorul cercetează directorul bibliotecii pentru numele de fişier.

Cînd are loc încărcarea unui fişier, el poate fi listat, sau modificat după cum cere aplicaţia specifică. Formatul:

LOAD *nume fişier*

unde:

nume fişier este numele unui fişier binar creat de către o comandă SAVE anterioară

Exemple:

LOAD "$PTR"
LOAD "MATH3"
LOAD "MTO:1"

Comanda **WHATS**

Scopul: Această comandă va imprima la teletype, informaţii legate de fişierul cu numele specific *nume fişier*. Tipul de informaţii imprimată şi formatul în care ea va fi redată pe imprimantă este:

nume fişier atribute lungimea (BYTES) data creerii data ultimei utilizări

Format:

WHATS *nume fişier*

unde: *nume fişier* este numele unui fişier curent de pe disc.

Exemplu:

WHATS "ABC"
ABC D 2039 03/17/75 (06/10/76)

Capitolul XIII

UTILIZAREA CALCULATORULUI NOVA ÎN REGIM DE CALCULATOR DE BIROU ŞI DEPANAREA DINAMICĂ A UNUI PROGRAM

Folosirea calculatorului Nova în regim de calculator de birou, este o componentă a modului de operare consolă (keyboard mode of operation). Aici termenul de „calculator de birou" este întrebuinţat în sensul că în compilatorul Extended BASIC, calculele se efectuează pe loc, obţinîndu-se imediat rezultatul numeric cerut. Folosirea acestui termen nu este cel mai adecvat, uneori aplicaţia „calculator de birou" (desk calculator) însemnînd operaţia prin care utilizatorul solicită executarea unui program introdus în acel moment în sistem prin intermediul terminalului sau prin care un program în prealabil catalogat într-un fişier de pe disc, este încărcat în memorie şi executat.

Aplicaţia denumită „depanarea dinamică a programelor" prin care utilizatorul cere ca sistemul să-i imprime anumite rezultate intermediare în scopul explicării erorilor întîlnite în timpul rulării, constituie alt aspect al utilizării calculatorului Nova în regim de calculator de birou [68].

Această ultimă problemă va fi tratată într-un subcapitol următor.

În exemplul de mai jos este ilustrat modul de folosire al calculatorului pentru calcularea expresiilor aritmetice care conţin doar elemente constante.

Se procedează la utilizarea comenzii PRINT (;) urmată de o expresie.

```
;EXP(SIN(3.4/8))1.51032
;USING "+####.##↑ ↑ ↑ ↑", EXP(SIN(3.4/8))+1510.32E−03
```

Apăsînd tasta RETURN sistemul va calcula imediat valoarea expresiei aritmetice şi o va imprima pe aceeaşi linie. Exemplele arată expresii care conţin operanzi literali.

Utilizatorul poate să includă, în afară de operanzi literali, şi valori atribuite variabilelor de program, el putînd să întrerupă rularea unui program prin apăsarea tastei ESC. În continuare el va putea să ceară înainte de reluarea execuţiei programului imprimarea unor şiruri de rezultate [61], [63], [65].

```
0010 DIM A$ [10], B$ [10]
0020 LET A$="YOU $10.50"
0030 B$="XRAY"
RUN
(ESC)
;B$(4); A$(2, 3) YOU
```

A fost cerută valoarea şirului format din concatenarea şirurilor $B\$(4)$ şi $A\$(2, 3)$. Rezultatul se va obţine pe aceeaşi linie şi anume YOU.

```
0010 DIM A [3, 3]
:
RUN
(ESC)
STOP AT 0500
```

Sistemul va indica înaintea cărei linii de program se va opri execuţia

;USING "+ #. #####E ↑ ↑ ↑ ↑ ", A(1, 2), A(1, 2)*9+5.12100E+02+ +4.68898E+01

:

13.1. DEPANAREA DINAMICĂ A UNUI PROGRAM

Un program în curs de rulare, poate fi întrerupt prin utilizarea instrucţiunii ESC sau prin executarea instrucţiunii STOP programat, într-un număr diferit de puncte. Sînt testate apoi valorile variabilelor curente în acele puncte şi se fac modificările de rigoare, atît la enunţuri cît şi la variabile, introducîndu-se chiar variabile şi enunţuri noi.

Programatorul va putea relua rularea programului întrerupt utilizînd comanda RUN, fără a pierde vreuna din valorile variabilelor din punctul de întrerupere.

În scopul facilizării depanării dinamice a unui program, sistemul de operare RDOS al calculatorului NOVA este prevăzut cu o componentă specifică numită *Symbolic Debugger* care va fi în continuare notată cu acronimul SD. (Depanator Simbolic).

13.1.1. FOLOSIREA, VERSIUNILE ŞI FORMATUL COMENZILOR DEPANATORULUI SIMBOLIC

Prin depanare, se înţelege procesul de detectare, localizare şi înlăturare a erorilor de naturi diferite, dintr-un program. Cînd se doreşte ca un program să fie depanat, depanatorul simbolic va fi încărcat împreună cu programul. În continuare utilizatorul trebuie să controleze execuţia programului, provocînd oprirea în depanator în unul sau mai multe puncte, astfel încît programatorul să poată examina conţinutul locaţiilor de memorie şi al registrelor speciale (acumulatoarele şi Carry) şi să poată corecta conţinutul lor dacă poate fi examinat în format sursă, putînd fi folosit şi formatul octal [65], [68].

Starea maşinii poate fi monitorizată în timpul execuţiei folosind comenzi simple pentru SD date de la teletype (TT).

Depanatorul Simbolic DGC*, permite programatorului să prevadă pînă la 8 puncte de întrerupere într-un program. Cînd este rulat, un program, execuţia sa se va opri înainte de a fi executată instrucţiunea din punctul de întrerupere, programatorul putînd astfel să folosească comenzile depanatorului. Apoi va fi reluată execuţia de la punctul de întrerupere sau de la orice altă locaţie se doreşte.

---

* DGC — Data General Corporation.

Pentru sistemul de operare RDOS, există două versiuni de SD:

— DEBUG III care permite toate întreruperile în timpul depanării.

— RDOS IDEB care interzice toate întreruperile. Dacă utilizatorul are un sistem RDOS cu MMPU*, atunci IDEB permite producerea tuturor întreruperilor cu excepţia acelora care sînt esenţiale pentru utilizarea în legătură cu facilitatea de „mapping". Ambele versiuni sînt sub formă binară relocabilă şi se găsesc în biblioteca RDOS.

O comandă pentru SD, are forma generală:

[*argument*] [$] *codul de comandă*

unde *codul de comandă* este un singur caracter al teletyp-ului.

$ trebuie să preceadă toate codurile de comandă alfabetice. El se pune înaintea anumitor coduri simbolice de comandă.

argument poate avea una din următoarele semnificaţii:

sim simbolul utilizatorului

adr orice adresă avînd un format valid de adresă: întreg octal sau zecimal, simbolul utilizatorului sau o expresie de forma $x \pm x \pm x \pm \ldots$ unde fiecare x este un simbol de utilizator sau un întreg în octal sau în zecimal.
Întregii zecimali, pentru a putea fi distinşi de cei în octal, trebuie urmaţi de un punct zecimal.

n un întreg zecimal sau octal

nume un simbol de utilizator care numeşte un program

adr< o gamă de adrese de la *adr* la 777777

adr<*adr*$_n$ o gamă de adrese de la *adr* la *adr*$_n$.

Comenzile care vor fi tratate în paragrafele următoare, prevăd facilităţi pentru:

— stabilirea, ştergerea şi examinarea punctelor de întrerupere
— reluarea execuţiei din punctele selectate
— monitorizarea memoriei, acumulatoarelor şi registrelor speciale
— plasarea programului de monitorizare într-un fişier care poate fi salvat
— perforarea sau imprimarea unor porţiuni din programul utilizatorului.

13.2. CONVENŢII ŞI SIMBOLURI ÎN COMANDA LINIILOR TT*

↵ Apăsarea tastei RETURN este reprezentată prin simbolul ↵. Nu se va imprima nimic la imprimanta TT, atunci cînd este apăsată tasta RETURN.

↓ Apăsarea tastei LINE FEED este reprezentată cu simbolul ↓, această acţiune nefiind urmată de imprimarea vreunui simbol pe imprimanta TT.

↑ Acest simbol va fi imprimat, dacă se apasă simultan tastele SHIFT şi N.

$ Dacă se acţionează tasta ESC, va fi imprimată simbolul $. Acelaşi rezultat se obţine prin acţionarea simultană a tastelor SHIFT şi 4.

? Apăsînd RUBOUT, va fi imprimat simbolul ?. Prin acţionarea acestei taste, SD va ignora comanda curentă de linie şi va imprima un ?

* MMPU — Memory Management and Protection Unit.
** TT — teletype.

În continuare programatorul va trebui să dactilografieze o nouă comandă de linie. Poate fi folosit orice caracter care provoacă o comandă de linie ilegală, întrucît modificarea acestei stări prin apăsarea tastei RUBOUT este foarte convenabilă.

13.3. RĂSPUNSUL ERORILOR COMISE ÎN COMENZILE DE SD

Dacă se încearcă folosirea unui simbol nedefinit într-o comandă a depanatorului, se va tipări un „U" după comanda eronată. Astfel

STAR/U — Utilizatorul face referire la locaţia simbolică STAR. Simbolul STAR nu este găsit în program, iar SD va răspunde cu un U aşteptînd o nouă comandă.

Toate celelalte erori de comandă vor avea ca răspuns „?" tipărit imediat în urma comenzii, după cum se poate vedea în exemplele următoare.

ADD@? — o terminare improprie a comenzii

—$R — o adresă ilegală care precede $R

START+6$B — Aceasta constituie o încercare de a poziţiona un punct de întrerupere la locaţia simbolică START+6. Va fi imprimat un răspuns eronat „?", dacă în program există deja 8 puncte de întrerupere.

Toate versiunile de SD înhibă întreruperile, cu excepţia RDOS Debug III, care operează cu un monitor „single task" ce permite întotdeauna producerea întreruperilor. Singurul mod în care poate fi întrerupt SD cînd se utilizează RDDS Debug III, este realizat prin apăsarea tastelor CTRL şi A simultan. În urma acestei acţiuni se va abandona activitatea programului curent.

13.4. COMENZI DE MONITORIZARE A MEMORIEI ŞI A REGISTRELOR SPECIALE

13.4.1. MONITORIZAREA MEMORIEI

În programul utilizatorului, locaţiile de memorie pot fi selectate, examinate şi modificate folosind una din comenzile următoare:

$\overline{adr}$/ selectează $\overline{adr}$ şi imprimă conţinutul adresei.

$\overline{adr}$! selectează $\overline{adr}$ şi nu îi imprimă conţinutul.

Mai sus prin $\overline{adr}$ se înţelege orice expresie acceptată de a defini o locaţie. Atunci cînd locaţia de memorie a fost selectată, folosind comanda / sau !, utilizatorul trebuie să modifice conţinutul locaţiei şi să o închidă (deselecteze), sau el trebuie să închidă locaţia fără modificări.

Utilizatorul modifică conţinutul locaţiei prin tipărirea noului conţinut pe aceeaşi linie cu comanda care selectează locaţia.

Utilizatorul poate să închidă locaţia selectată într-unul din următoarele feluri:

↵ Închide locaţia selectată (Cheia RETURN)

↓ Închide locaţia selectată şi selectează locaţia următoare (Cheia LINE FEED)

↑ Închide locaţia selectată şi selectează locaţia precedentă (Cheile SHIFT şi N)

/or ! Închide locaţia selectată şi selectează locaţia specificată prin conţinutul acelei locaţii.

În continuare vor fi prezentate cîteva exemple de închidere şi selectare de locaţii.

START/ 006011 6017 ↑ +762 000000 ↑ +761 177400 ↑ +760 177400 ↑ ↵	Se selectează locaţia simbolică START şi se modifică conţinutul lui 6017. Se selectează locaţia precedentă şi nu se modifică. Se închide locaţia şi se selectează locaţia precedentă. Se închide locaţia.
START/006011 6017 +762 000000 +761 177400 +760 177400	La consolă, exemplul va apare precum urmează.
START / 006017 ↓	Se selectează locaţia START şi locaţia următoare
START+1 001400 ↓	Se selectează locaţia următoare
START+2 006073 ↓ ↵	Se selectează locaţia următoare şi se închide

La consolă va apare scris astfel:

START / 006017
START+1 001400
START+2 006073

1000/.RDL O.RDL 1 ↓ ↵	Deschide locaţia 1000 şi schimbă conţinutul la .RDL 1. Închide locaţia
1000 !. RDL 0 ↓ ↵	Deschide locaţia 1000 fără să îi imprime conţinutul. Schimbă înapoi conţinutul lui .RDL 0.
1000/ .RDL 0. RDL 1.	Pe imprimanta TT, exemplele vor apare astfel: 1000 !. RDL 0.

13.4.2. MONITORIZAREA REGISTRELOR SPECIALE

Registrele speciale sînt locaţii conţinînd informaţii de stare a programului. Ele pot fi selectate, examinate, modificate şi închise într-un mod similar cu locaţiile de memorie. Un registru special este în mod normal închis, prin apăsarea tastei RETURN sau utilizînd convenţia „/or !".

Acumulatoarele

Comanda care deschide un acumulator pentru examinare şi o posibilă modificare, este:

n $A

unde *n* este numărul acumulatorului (0—3).

Exemplu: 0 $A 0000 1

Utilizînd comanda

$A

toate cele 4 acumulatoare vor fi examinate dar nu vor fi modificate, SD va realiza o întoarcere a carului de imprimat şi va imprima numărul fiecărui acumulator urmat de conţinutul lui.

Exemplu: $A

0 000000 1 000654 2 001198 3 054673

Registrul de numere

Registrul de numere determină dacă conţinutul registrelor va fi imprimat în octal sau în zecimal. În mod standard registrul de numere este pus pe 0, provocînd imprimarea conţinutului registrelor în octal. Dacă utilizatorul pune conţinutul registrului de numere pe o cifră diferită de zero, conţinutul memoriei şi al registrelor generale va fi imprimat în zecimal, în convenţia numere zecimale cu semn cu punct zecimal.

Registrul de numere este deschis pentru examinare şi modificare, utilizînd comanda:

$*N*

Exemplu:

START/006017 — conţinutul lui START în octal
$N 000000 1 — modifică conţinutul registrului de numere pentru a fi diferit de zero
START /+3087 — conţinutul lui START în zecimal
$N +1.0 — conţinutul registrului de numere zecimal

Registrul blocului de control al sarcinii (taskului)

Registrul blocului de control al sarcinii conţine adresa blocului de control al sarcinii (BCT) executat curent. BCT conţine informaţiile de stare pentru fiecare sarcină de care are nevoie planificatorul de sarcini (task scheduler) în modul de exploatare multisarcină. Acest registru este folosit doar de către RDOS Debug III. Registrul este deschis pentru examinare şi pentru posibile modificări, prin comanda:

$*T*

Exemplu:

$*T* 000654 — deschide BCT şi îi examinează conţinutul.

Registrul locaţiei de start

În sistemul de operare RDOS Debug III, registrul locaţiei de start, conţine adresa planificatorului de sarcini, permiţînd ca să fie transferat, controlul planificatorului (scheduler-ului) în scopul rulării taskului cu prioritatea cea mai ridicată. Registrul locaţiei de start poate fi selectat pentru examinare şi pentru o posibilă modificare prin comanda:

$*L*

Exemplu:

$*L* 000764 determină adresa task scheduler-ului (RDOS Debug III)

13.5. PUNCTE DE ÎNTRERUPERE ŞI RESTARTAREA PROGRAMULUI

Punctele de întrerupere sînt partea esenţială a procesului de depanare. Ele permit utilizatorului să execute o mică porţiune a programului său şi

apoi să verifice starea programului. Utilizatorul pune unul sau mai multe puncte de întrerupere în programul său, folosind o comandă de depanare. Apoi el poate să indice printr-o comandă cînd un punct de întrerupere ar produce încetarea execuţiei programului şi cînd va fi făcut transferul către depanator.

În momentul cînd este întîlnit un punct de întrerupere, instrucţiunea de program la care a fost pus acest punct de întrerupere este transferată depanatorului şi o instrucţiune de tip JMP* spre depanator este substituită în programul utilizatorului.

Sînt rezervate opt locaţii din pagina zero, pentru cele 8 puncte de întrerupere permise. Aceste puncte de întrerupere au atribuite valori numerice în ordine inversă a şirului numere naturale. Astfel dacă utilizatorul prevede 5 puncte de întrerupere în programul său, ele vor fi numărate: 7, 6, 5, 4, 3.

13.5.1. STABILIREA, EXAMINAREA ŞI ŞTERGEREA PUNCTELOR DE ÎNTRERUPERE

Pentru stabilirea unui punct de întrerupere se foloseşte comanda:

adr$*B*

unde:

adr este adresa programului la care se stabileşte punctul de întrerupere

Exemplu:

START$*B*
START+72$*B*
START+86$*B*

Nu pot fi prevăzute puncte de întrerupere la următoarele tipuri de locaţii; în cazul datelor, în cazul instrucţiunilor care se modifică în timpul execuţiei programului şi pentru locaţiile unde întreruperile pot fi întîrziate pentru un timp relativ lung.

Este posibil ca utilizatorul să modifice punctele de întrerupere din locul unde sînt puse în mod curent în programul său. Pentru a imprima numărul de puncte întrerupere şi locaţiile la care sînt poziţionate se întrebuinţează comanda:

$*B*

După această comandă, punctele de întrerupere vor fi imprimate în ordine numerică descrescătoare.

Exemple:

$*B*
7B START
6B START +33
5B START +42

Pentru ştergerea unui singur punct de întrerupere se utilizează comanda:

n$*D*

unde *n* este numărul unui punct de întrerupere în prealabil stabilit. Comanda va şterge punctul de întrerupere specificat şi celelalte numere de întrerupere

* JOB MANAGEMENT PROGRAM.

vor rămîne neschimbate. Astfel dacă de exemplu au fost stabilite ca puncte de întrerupere punctele 7, 6, 5 comanda:

6$D

va şterge punctul de întrerupere 6, în timp ce numerele asignate punctelor de întrerupere care rămîn nu vor fi schimbate.

Pentru a şterge toate punctele de întrerupere dintr-un program, utilizatorul va putea utiliza comanda:

$D

13.5.2. NUMĂRĂTOARELE DE PUNCTE DE ÎNTRERUPERE

Fiecare punct de întrerupere are asociat cîte un numărător, care indică momentul în care va avea loc o comutare la depanator, în timpul execuţiei programului.

Atunci cînd este stabilit un punct de întrerupere, numărătorul pentru acel punct de întrerupere este pus în mod standard pe 1, indicîndu-se de cîte ori instrucţiunea de la punctul de întrerupere va fi executată înainte ca depanatorul să reintre în funcţiune.

Comanda pentru a deschide numărătorul de puncte de întrerupere, este:

n$*Q*

unde: *n* este numărul punctului de întrerupere poziţionat în prealabil

Exemplu:

7$*Q* 000001 2 Schimbă la două numărul de execuţii ale instrucţiunii la punctul de întrerupere 7, înaintea reintrării în depanator.

13.5.3. COMENZI DE RESTARTARE A PROGRAMULUI

Pentru restartarea programului depanat există comanda: $*R*

Programul se va executa, ciclînd printr-un punct de întrerupere, de numărul de ori indicat de numărătorul de întreruperi, pînă cînd este întîlnit un punct de întrerupere.

$*R*
7B START
0 006207 1 006162 2 000000 3 006162

Utilizatorul poate să specifice locaţia la care este stopată execuţia programului cu comanda:

adr$*R*

unde: *adr* este o adresă în interiorul programului.

START+2$*R*
6B START+10
0 001654 1 000000 2 000000 3 000000

În urma transferului comenzii la depanatorul SD, într-un punct de întrerupere, utilizatorul enunţă comenzile de depanare şi poate apoi să reia execuţia programului la punctul de întrerupere prin comanda:

$*P*

Programul se va executa, ciclînd prin punctul de întrerupere de un număr de ori indicat de numărul de întreruperi. Apoi va fi imprimat conţinutul registrelor şi punctul de întrerupere, în momentul cînd SD reintră.

$R
7B START
0 006207 1 6162 2 000004 3 006162
$P
6B START+10
0 001654 1 000000 2 000004 3 000000

13.6. SALVAREA UNUI PROGRAM DEPANAT

În depanatoarele RDOS, se prevede o posibilitate de salvare a stării curente a programului de depanare. Comanda $V permite programatorului să se reîntoarcă de la depanatorul SD, la nivelul CLI* cu starea curentă a programului depanat în fişierul BREAK, care include starea curentă a tuturor registrelor şi corecţiile făcute în timpul depanării. Utilizatorul în starea CLI, poate să ia orice acţiune potrivită acelui moment. Dacă se doreşte salvarea fişierului BREAK, el poate să emită o comandă CLI SAVE, care va salva fişierul sub numele dat. Exemplul de utilizare a comenzii $V:

DEB ALPHA	comenzi de depanare
$V	se reîntoarce la CLI
BREAK	
R	
R	
SAVE ALPHA	Programatorul salvează fişierul BREAK, sub numele ALPHA
R	

Înainte de emiterea comenzii $V, utilizatorul va şterge toate punctele de întrerupere (DS). Altfel, o încercare de a depana fişierul salvat va provoca o oprire. Comanda $V va face ca toate fişierele deschise să fie închise. Pentru execuţia fişierului de întreruperi după emiterea lui $V, utilizatorul trebuie să prevadă o rutină de redeschidere a tuturor fişierelor care erau deschise atunci cînd a fost emisă comanda $V.

13.7. ÎNCĂRCAREA LUI RDOS DEBUG III

RDOS Debug III este conţinut în biblioteca de benzi, SYS.LB şi este încărcat cu programul utilizatorului atunci cînd utilizatorul dă comutatorul global */D* pentru comanda RLDR, adică: RLDR/*D* ALPHA ↵.

Comanda încarcă ALPHA urmată de DEBUG şi adaugă tabela de simboluri la fişierul salvat ALPHA.SV imediat după ultima locaţie DEBUG care a fost încărcată [68].

---

* Command Line Interpreter.

SUMARUL COMENZILOR DEPANATORULUI SIMBOLIC RDOS DEBUG III*

Comanda	Tipul comenzii	Semnificaţia
adr !	memoria monitorului	Selectează locaţia de memorie *adr*
adr/		Selectează locaţia de memorie *adr* şi imprimă conţinutul
↵		Închide locaţia selectată
↓		Închide locaţia selectată şi deschide locaţia următoare
↑		Închide locaţia selectată şi deschide locaţia anterioară
\$A	acumulatoarele monitorului	Imprimă conţinutul tuturor acumulatoarelor
n\$A		Selectează acumulatorul *n* (*n*=0–3)
\$C	monitor special	Selectează registrul carry
\$L	registre	Selectează registrul de locaţie
\$M		Selectează registrul de mască
\$N		Selectează registrul de număr
\$T		Selectează registrul blocului de control al task-ului
\$W		Selectează registrul de cuvînt
\$Y		Selectează registrul de punctare a tabelei de simboluri
\$B	punct de întrerupere	Imprimă locaţia tuturor punctelor de întrerupere ale programului
adr\$B		Inserează punctul de întrerupere la locaţia *adr*
\$D		Şterge toate punctele de întrerupere
n\$D		Şterge punctul de întrerupere *n* (*n*=0–7)
\$V	fişier de întrerupere	Pune programul depanat în fişierul de întrerupere pentru posibila salvare a fişierului
\$P	execută (şi numărătorul de întreruperi)	Reia execuţia de la un punct de întrerupere cu numărătorul de întreruperi egal cu 1
n\$Q		Deschide numărătorul de întreruperi *n* (*n*=0–7)
\$R		Restartează execuţia la adresa din USTSA**
n\$P		Reia execuţia de la un punct de întrerupere cu numărătorul de întreruperi pus pe *n*
adr\$R		Restartează execuţia la *adr*
\$K	simboluri admise şi interzise	Mută toate simbolurile locale şi globale de la intrare şi ieşire

* Vezi [68].
** Adresa task scheduler-ului.

Comanda	Tipul comenzii	Semnificaţia
n$K		Mută toate simbolurile de la intrare/ieşire dar reţine pe cele globale
sym$K		Mută simbolul *sym* de la ieşire în mod permanent
name %		permite toate simbolurile locale şi globale în programul *name*
$S	căutarea memoriei	Cercetează toată memoria
adr$S		Cercetează memoria de la locaţia zero la *adr*
adr1<$S		Cercetează memoria de la locaţia *adr1* la limita memoriei
adr1<*adr2* $S		Cercetează memoria de la locaţia *adr1* la locaţia *adr2* inclusiv
=	formatul datelor de ieşire la teletype	Imprimă ultima dată tipărită în format numeric
:		Imprimă ultima dată tipărită în format simbolic
;		Imprimă ultima dată tipărită în format instrucţiune
←		Imprimă ultima dată tipărită în format de semicuvînt
,		Imprimă ultima dată tipărită în formatul ASCII
&		Imprimă ultima dată tipărită în format de punctator de byte
$=		Imprimă data ulterioară în format numeric
$:		Imprimă data ulterioară în format simbolic
$;		Imprimă data ulterioară în format instrucţiune
$←		Imprimă data ulterioară în format semicuvînt
$'		Imprimă data ulterioară în format ASCII
$&		Imprimă data ulterioară în format de punctator de byte
?		Imprimă ultima dată tipărită în .SYSTM în format de comandă
$?		Imprimă data ulterioară în format de comandă .SYSTM

În acest exemplu programul depanator este încărcat după programul utilizatorului. Utilizatorul poate să controleze momentul încărcării depanatorului *SD*.

RLDR/*D A B* SYS.LB *C D*↵

Se observă că depanatorul Debug III este încărcat după B şi înainte de încărcarea lui *C*.

De îndată ce depanatorul corespunzător sistemului de operare folosit a fost încărcat, el este chemat prin emiterea comenzii DEB urmată de numele fişierului salvat.

Primul nume de fişier relocabil binar care apare în comanda de linie RLDR, este în mod standard numele atribuit fişierului de salvare.

RLDR/*D A B C*↵
DEB *A*↵

Cînd este dată comanda *DEB*, depanatorul *SD* va răspunde cu o reîntoarcere a carului, şi utilizatorul poate din nou să procedeze la emiterea de comenzi către depanator.

13.8. EXEMPLE DE UTILIZARE A PROGRAMULUI DE DEPANARE DINAMICĂ

Un program care se rulează, poate fi, după cum se cunoaşte, întrerupt într-un număr diferit de puncte de program, utilizînd instrucţiunile ESC sau STOP programat. În continuare, valorile curente ale variabilelor pot fi verificate în aceste puncte şi pot fi făcute corecţiile în program, atît la instrucţiuni cît şi la variabile dacă este necesar. Programatorul poate să utilizeze apoi comanda RUN *nici-o instrucţiune* pentru a relua programul întrerupt, fără pierderea vreuneia din valorile variabilelor în punctul de întrerupere sau a vreuneia din noile valori şi instrucţiuni inserate.

⋮	
(ESC)	
STOP *AT* 1100	
IF *A*<>*B* THEN PRINT *B*, *A*↵	Comanda de utilizator prevăzută în mod condiţional pentru examinarea lui *A* şi *B*
.094 .9	
⋮	
2.33333 6.47653 9.67584	rezultatele unei serii de calcule de program au fost imprimate
(ESC)	
STOP AT 0760	
READ *X*1, *X*2, *X*3 ↵ RUN 760 ↵	Utilizatorul sare peste următoarele 3 valori din blocul de date şi reîncepe execuţia programului de la instrucţiunea la care a fost întrerupt

(ESC)	
STOP *AT* 1100	
; A 0	Utilizatorul verifică valoarea variabilei *A*
A=−1 ↵	Utilizatorul schimbă şirul de variabile
C$="% OF LOSS" ↵	*C*$ şi valoarea variabilei aritmetice *A*
RUN 505	şi reia rularea la instrucţiunea 505
⋮	
20 DIM *A* [4, 4]	
⋮	
(ESC)	
STOP AT 500	
DIM *A* [3, 5]	Utilizatorul redimensionează zona *A*.

Capitolul XIV

14. ALGORITMI PENTRU REZOLVAREA ECUAȚIILOR ALGEBRICE NELINIARE

Din literatură [58] rezultă că există o strînsă legătură între matematică și calculatoare în scopul formulării și rezolvării problemelor. Toate problemele necesită concepte matematice pentru formularea lor, iar tehnicile de calcul de asemenea necesită o serie de concepte pentru rezolvarea practică a problemelor.

Tehnicile de calcul moderne introduc o serie de simplificări în formularea și rezolvarea problemelor cu ajutorul calculatoarelor electronice.

Familiarizarea cu aceste tehnici este esențială nu numai pentru cei ce lucrează în domeniul calculatoarelor, sau a celor ce utilizează calculatoarele ci și pentru cei care doresc să înțeleagă cum un calculator execută o serie de activități destul de dificile. Se poate afirma că în domeniul matematicii au apărut o serie de preocupări noi și a crescut numărul celor care fac cercetări în domeniul matematicilor moderne dar este destul de mic numărul celor care utilizează matematica și tehnicile de calcul electronic în cercetările lor proprii.

Calculatorul este implicat atît în procesul de formalizare a problemei cît și în procesul de rezolvare în mod direct. În continuare se vor prezenta o serie de metode pentru abordarea cu ajutorul calculatorului a unor modele matematice.

În o serie de aplicații practice se ajunge în final la găsirea zerourilor unei funcții de forma

$$\begin{aligned} f(x)&=\sin x-x+2=0\\ f(x)&=x^3-5=0\\ f(x)&=e^x+2\ln x-4=0 \end{aligned} \tag{1}$$

Toate aceste funcții sînt funcții reale de variabilă reală în plus se presupune că $f(x)$ este continuă. Dacă $f(\alpha)=0$ atunci, α este o rădăcină a lui $f(x)$. În literatură [12] sînt prezentate o serie de metode aproximative pentru determinarea zerourilor ecuațiilor algebrice de tipul (1). Dintre aceste metode se pot enunța: metodele grafice, metoda bisecției, metoda poziției false, metoda secantei, metoda Newton, metoda Muller etc.

În cazul în care se utilizează o metodă grafică pentru găsirea soluției unei ecuații de forma

$$f(x)=\cos x-2x+3=0 \tag{2}$$

se construiește graficul lui $f(x)$ și se caută punctele unde acest grafic intersectează axa Ox, aceste puncte reprezintă rădăcinile reale ale lui $f(x)$.

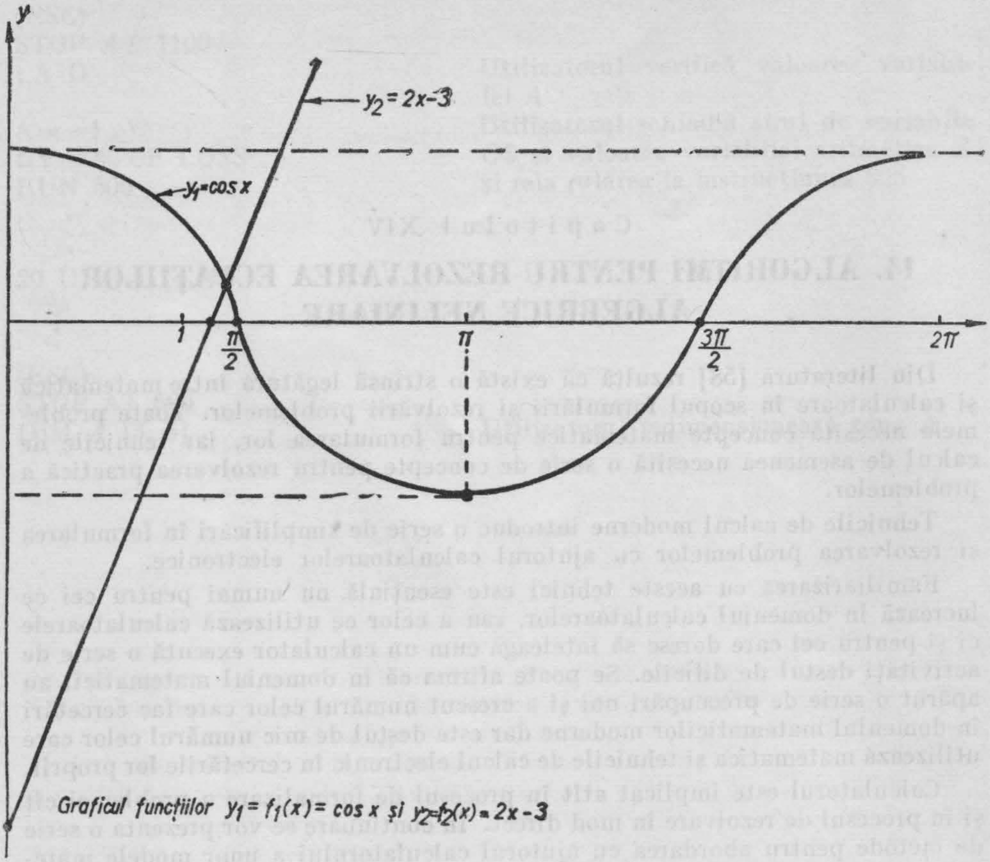


Fig. 14.1

De obicei în cazul (2) este convenabil a se trasa două grafice y_1 și y_2, adică se scrie funcția $f(x)$ sub forma diferenței a două funcții $f_1(x)$ și $f_2(x)$ astfel ca

$$f(x)=f_1(x)-f_2(x),\ f_1(x)=\cos x=y_1,\ f_2(x)=2\ x-3=y_2 \tag{3}$$

În urma acestei transformări apare evident faptul că $f(x)=0$ dacă și numai dacă $y_1=y_2$.

Metoda constă în trasarea graficelor funcțiilor $f_1(x)$ și $f_2(x)$ și determinarea punctului (x, y) unde cele două grafice se intersectează fig 14.1. Valoarea lui x corespunzătoare punctului de intersecție a celor două grafice va fi rădăcina reală a funcției (2). Avantajul metodei care necesită trasarea a două grafice este evidentă prin faptul că în foarte multe cazuri este mult mai simplu să trasezi graficele pentru $f_1(x)$ și $f_2(x)$ decît pentru $f(x)$. Rădăcina aproximativă obținută prin metode grafice poate fi utilizată ca valoare de start pentru o metodă iterativă care conduce după un număr de iterații la o soluție mult îmbunătățită.

```
0001 REM PRØGRAM PI CAP.14.
0003 REM PRØGRAM PENTRU LØCALIZAREA CU AJUTØRUL METØDEI GRAFICE
0005 REM A RADACINILØR REALE ALE FUNCTIEI
0010 DEF FNF(X)=CØS(X)
0020 DEF FNG(X)=2*X-3
0030 INPUT A,B,H1,H2
0031 INPUT H3
0040 LET D=(B-A)/H1
0050 DIM V[D,1],W[D,1]
0055 LET J=0
0060 FØR I=A TØ B STEP H1
0070   LET V[J,1]=FNF(I)/H2
0080   LET W[J,1]=(FNG(I))
0081   LET W[J,1]=(W[J,1])/H3
0090   LET J=J+1
0100 NEXT I
0110 LET M1=V[0,1]
0120 LET M2=W[0,1]
0130 FØR I=A TØ D
0140   IF M1<=V[I,1] THEN GØTØ 0160
0150   LET M1=V[I,1]
0160   IF M2<=W[I,1] THEN GØTØ 0180
0170   LET M2=W[I,1]

0180 NEXT I
0190 IF M1>=0 THEN GØTØ 0240
0200 IF M2>=0 THEN GØTØ 0220
0210 IF M2<=M1 THEN GØTØ 0250
0220 LET M3=-M1
0230 GØTØ 0270
0240 IF M2>=0 THEN GØTØ 0230
0250 LET M3=-M2
0260 GØTØ 0230
0270 FØR I=A TØ D
0280   IF I<>0 THEN GØTØ 0400
0290   FØR J=0 TØ 70
0300     LET S=INT(W[0,1]+M3)
0310     IF J<>S THEN GØTØ 0340
0320     PRINT "@";
0330     GØTØ 0390
0340     LET S=INT(V[0,1]+M3)
0350     IF J<>S THEN GØTØ 0330
0360     PRINT "*";
0370     GØTØ 0390
0380     PRINT "-";
0390   NEXT J
0393   PRINT
0395   GØTØ 0630
0400   IF INT(W[I,1]+M3)<>INT(M3) THEN GØTØ 0440
0420   PRINT  TAB(M3)"0"; TAB(V[I,1]+M3)"*"
0430   GØTØ 0630
0440   IF (W[I,1]+M3)>M3 THEN GØTØ 0500
0460   PRINT  TAB(W[I,1]+M3)"6"; TAB(M3)"!"; TAB(V[I,1]+M3)"*"
0470   GØTØ 0630

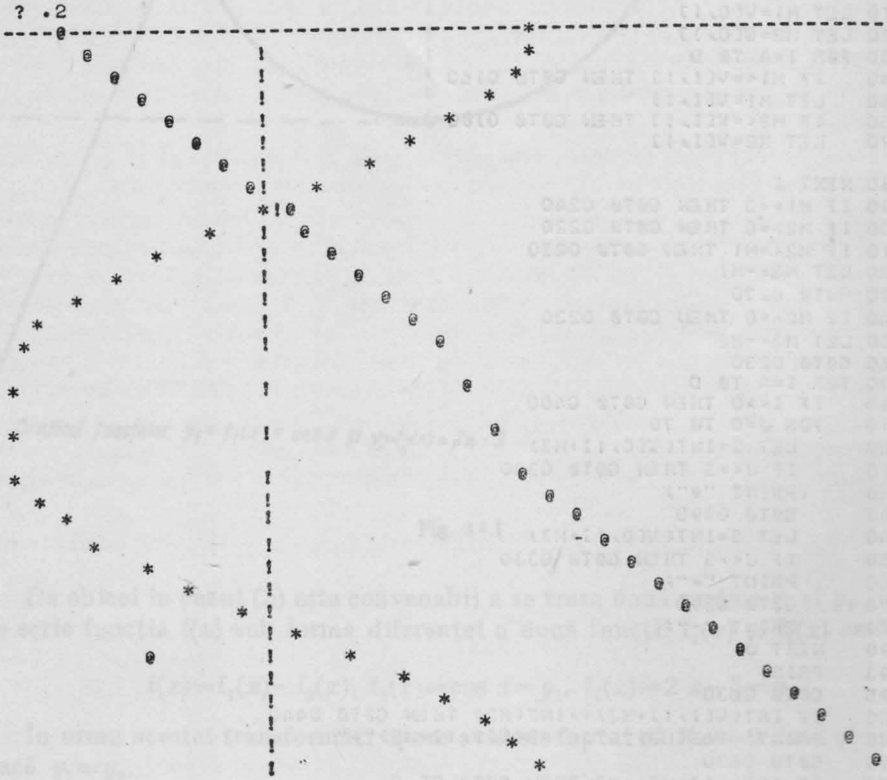
0500   IF INT(V[I,1]+M3)<>INT(W[I,1]+M3) THEN GØTØ 0530
0510   PRINT  TAB(M3)"!"; TAB(V[I,1]+M3)"A"

0520   GØTØ 0630

0530   IF INT(W[I,1]+M3)>INT(V[I,1]+M3) THEN GØTØ 0560
0540   PRINT  TAB(M3)"!"; TAB(W[I,1]+M3)"@"; TAB(V[I,1]+M3)"*"
0550   GØTØ 0630
0560   IF INT(V[I,1]+M3)<>M3 THEN GØTØ 0590
0570   PRINT  TAB(M3)"B"; TAB(W[I,1]+M3)"X"
```

```
0580   GØTØ 0630
0590   IF INT(V[I,1]+M3)<M3 THEN GØTØ 0620
0600   PRINT  TAB(M3)"!"; TAB(V[I,1]+M3)"*"; TAB(W[I,1]+M3)"@"
0610   GØTØ 0630
0620   PRINT  TAB(V[I,1]+M3)"*"; TAB(M3)"!"; TAB(W[I,1]+M3)"@"
0630 NEXT I
0640 END
```

```
* RUN
 ? 0 ? 6 ? .2 ? .05
 ? .2
```



```
END AT 0640
*
```

În continuare se va prezenta un program în BASIC care localizează cu ajutorul metodei grafice rădăcinile reale ale funcţiei

$$f(x)=e^x+2 \ln x-4=e^x-(4-2 \ln x)=f_1(x)-f_2(x)=0$$

prin construcţia a două grafice

$$f_1(x)=e^x \text{ şi } f_2(x)=4-2 \ln x$$

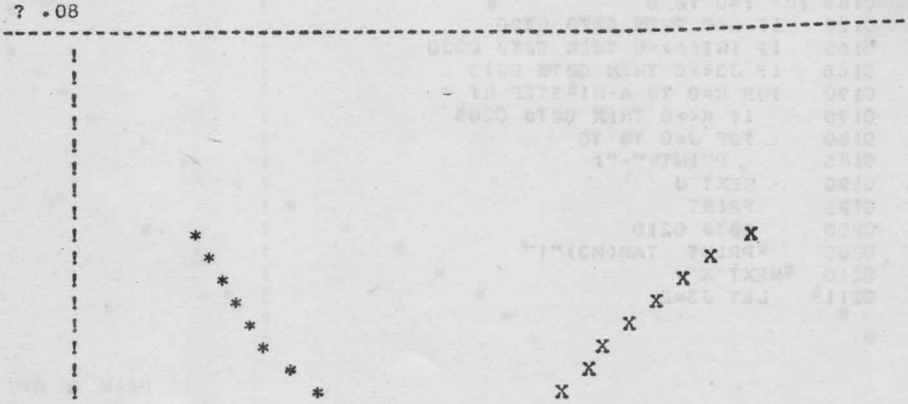
folosind acelaşi sistem de axe.

```
0001 REM PRØGRAM PENTRU GASIREA UNUI ZERØ PRIN METØDA GRAFICA
0002 REM PENTRU FUNCTIA F(X)=E↑X+2*LN(X)-4
0005 DEF FNF(X)=EXP(X)
0010 DEF FNG(X)=4-2*LØG(X)
0015 INPUT A,B,H1,H2
0020 INPUT H3
0025 LET D=(B-A)/H1
0030 DIM V[D,1],W[D,1]
0035 LET J=0
0040 LET J2=0
0041 LET J3=0
0045 FØR I=A TØ B STEP H1
0050   LET V[J,1]=FNF(I)/H2
0055   LET W[J,1]=FNG(I)
0060   LET W[J,1]=(W[J,1])/H3
0065   LET J=J+1
0070 NEXT I
0075 LET M1=V[0,1]
0080 LET M2=W[0,1]
0085 FØR I=0 TØ D
0090   IF M1<=V[I,1] THEN GØTØ 0100
0095   LET M1=V[I,1]
0100   IF M2<=W[I,1] THEN GØTØ 0110
0105   LET M2=W[I,1]
0110 NEXT I
0115 IF M1>=0 THEN GØTØ 0140
0120 IF M2>=0 THEN GØTØ 0130
0125 IF M2<=M1 THEN GØTØ 0145
0130 LET M3=-M1
0135 GØTØ 0155
0140 IF M2>=0 THEN GØTØ 0152
0145 LET M3=-M2
0150 GØTØ 0135
0152 LET M3=5
0155 FØR I=0 TØ D
0160   IF A<0 THEN GØTØ 0220
0165   IF INT(A)=0 THEN GØTØ 0230
0166   IF J3<>0 THEN GØTØ 0315
0170   FØR K=0 TØ A-H1 STEP H1
0175     IF K<>0 THEN GØTØ 0205
0180     FØR J=0 TØ 70
0185       PRINT "-";
0190     NEXT J
0195     PRINT
0200     GØTØ 0210
0205     PRINT  TAB(M3)"!"
0210   NEXT K
0211    LET J3=2
0215   GØTØ 0315
0220   LET A=A+H1
0221   LET J2=J2+1
0225   GØTØ 0315
0227   LET J2=J2+1
0230   IF J2=0 THEN GØTØ 0240
0235   GØTØ 0245
0240   IF I<>0 THEN GØTØ 0315
0245   FØR J=0 TØ 70
0255     IF J<>INT(W[I,1]+M3) THEN GØTØ 0275
```

```
0260      PRINT "X";
0265      GØTØ 0295
0275      IF J<>INT(V[I,1]+M3) THEN GØTØ 0290
0280      PRINT "*";
0285      GØTØ 0295
0290      PRINT "-";
0295    NEXT J
0300    LET J2=0
0305    PRINT
0310    GØTØ 0410
0315    IF INT(W[I,1]+M3)<>INT(M3) THEN GØTØ 0330
0320    PRINT  TAB(M3)"O"; TAB(V[I,1]+M3)"*"
0325    GØTØ 0410
0330    IF (W[I,1]+M3)>M3 THEN GØTØ 0345
0335    PRINT  TAB(W[I,1]+M3)"X"; TAB(M3)"!"; TAB(V[I,1]+M3)"*"
0340    GØTØ 0410
0345    IF INT(V[I,1]+M3)<>INT(W[I,1]+M3) THEN GØTØ 0360
0350    PRINT  TAB(M3)"!"; TAB(V[I,1]+M3)"A"
0355    GØTØ 0410
0360    IF INT(W[I,1]+M3)>INT(V[I,1]+M3) THEN GØTØ 0375
0365    PRINT  TAB(M3)"!"; TAB(W[I,1]+M3)"X"; TAB(V[I,1]+M3)"*"
0370    GØTØ 0410
0375    IF INT(V[I,1]+M3)<>INT(M3) THEN GØTØ 0390
0380    PRINT  TAB(M3)"B"; TAB(W[I,1]+M3)"X"
0385    GØTØ 0410
0390    IF INT(V[I,1]+M3)<M3 THEN GØTØ 0405
0395    PRINT  TAB(M3)"!"; TAB(V[I,1]+M3)"*"; TAB(W[I,1]+M3)"X"
0400    GØTØ 0410
0405    PRINT  TAB(V[I,1]+M3)"*"; TAB(M3)"!"; TAB(W[I,1]+M3)"X"
0410 NEXT I
0415 END

* RUN
*RUN

 ? 1,3,.1,.3
 ? .08
```



```
END AT 0415.
*
```

14.1. METODA ÎNJUMĂTĂŢIRII INTERVALULUI

Această metodă constă în determinarea unui zerou al funcţiei $f(x)$ cuprins între a şi b. Este bine cunoscut faptul că dacă există un număr $\alpha \in (a, b)$ astfel că $f(\alpha)=0$, şi $f(x)$ este continuă pe $[a, b]$ are loc relaţia

$$f(a)f(b)<0.$$

Metoda înjumătăţirii intervalului constă în împărţirea intervalului (a, b) în două părţi egale după care se testează care din cele două intervale conţine rădăcina. Algoritmul de calcul constă în

$a_0=a$, $b_0=b$, după care se calculează o estimaţie iniţială

$$c_0=\frac{a_0+b_0}{2}=\frac{a+b}{2}$$

Dacă $f(c_0)=0$ procesul este terminat, altfel se fac notaţiile:

$$a_1=a_0 \text{ şi } b_1=c_0 \text{ dacă } f(c_0)f(a_0)<0$$

sau

$$a_1=c_0, \quad b_1=b_0 \text{ dacă } f(c_0)f(a_0)>0 \tag{4}$$

În general, după n iteraţii se obţine a_n şi b_n astfel că

$f(a_n)\,f(b_n)<0$, după care se calculează

$$c_n=\frac{a_n+b_n}{2}$$

Dacă $f(c_n)=0$ procesul de calcul este terminat şi c_n este rădăcina căutată, altfel

$$\left.\begin{array}{l} a_{n+1}=a_n \\ b_{n+1}=c_n \end{array}\right\} \text{ dacă } f(c_n)\,f(a_n)<0$$

sau

$$\left.\begin{array}{l} a_{n+1}=c_n \\ b_{n+1}=b_n \end{array}\right\} \text{ dacă } f(c_n)\,f(a_n)>0$$

Convergenţa acestui proces se poate demonstra destul de uşor, deoarece şirul $a_0, a_1, a_2, \ldots$ este un şir crescător mărginit superior, iar şirul $b_0, b_1, b_2, \ldots$ este un şir descrescător mărginit inferior. În concluzie cele două şiruri converg.

Fie α şi β limita celor două şiruri a_n şi b_n respectiv Deoarece $a_n<c_n<b_n$, atunci [47]

$$\lim_{n\to\infty}|a_n-c_n|=0$$

de unde rezultă că $\alpha=\beta$ şi

$$\alpha=\lim_{n\to\infty} b_n=\lim_{n\to\infty} c_n=\lim_{n\to\infty} a_n$$

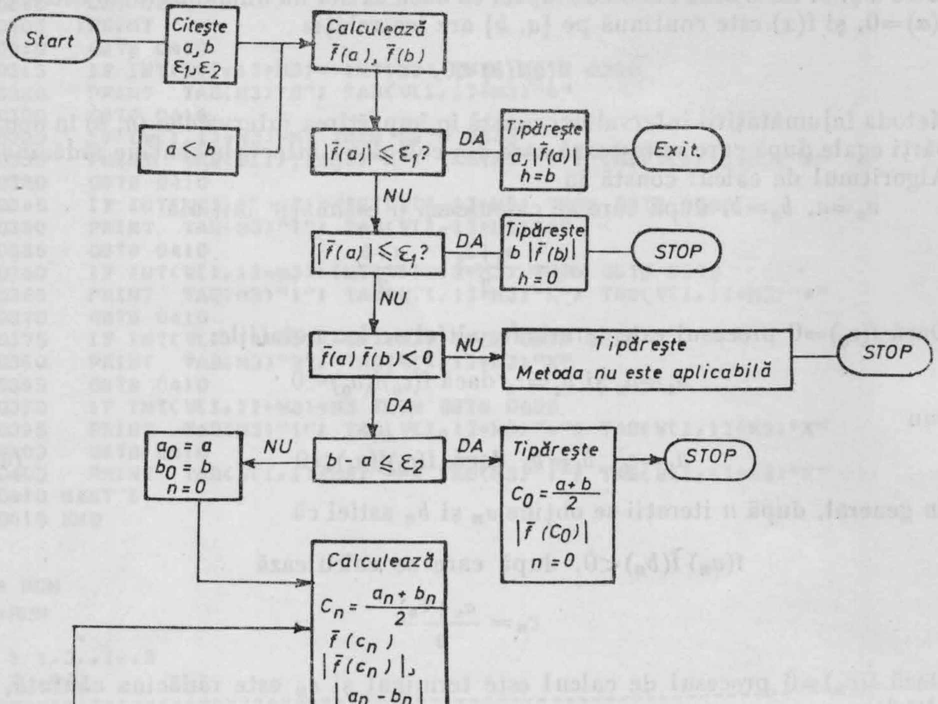

Start
Citeşte
a, b
ε1, ε2
a < b
Calculează
f̄(a), f̄(b)
|f̄(a)| ⩽ ε1?
DA
Tipăreşte
a, |f̄(a)|
h=b
Exit.
NU
|f̄(a)| ⩽ ε1?
DA
Tipăreşte
b |f̄(b)|
h=0
STOP
NU
f̄(a) f̄(b) < 0
NU
Tipăreşte
Metoda nu este aplicabilă
STOP
DA
b - a ⩽ ε2
DA
Tipăreşte
C0 = (a+b)/2
|f̄ (C0)|
n=0
STOP
NU
a0 = a
b0 = b
n=0
Calculează
Cn = (an + bn)/2
f̄ (cn)
|f̄ (cn)|,
|an - bn|
|f̄(cn)| ⩽ ε1?
DA
Scrie
Cn, n
|f̄ (cn)|
|bn - an|
STOP
NU
|bn - an| ⩽ ε2
DA
NU
n+1 → n
f̄(an) f̄ (cn) < 0
NU
cn → an + 1
bn → bn + 1
DA
an → bn + 1
cn → bn + 1

Fig. 14.2

Datorită faptului că

$$f(a_n)f(b_n)<0$$

pentru orice n rezultă că
$0 \geqslant \lim_{n\to 8} [f(a_n)f(b_n)] = (\lim f(a_n))(\lim f(b_n)) = [f(\alpha^2)]$, dar $[f(x)]^2 \geqslant 0$, de unde rezultă că $f(\alpha)=0$ și $\lim_{n\to\infty} f(a_n)=0$ datorită continuității lui $f(x)$.

În practică se lucrează cu valoarea lui $f(x)$ notată prin $\overline{f}(x)$. Pentru determinarea rădăcinei α a funcției $f(x)$ pe intervalul $[a, b]$ se introduc două constante pozitive ε_1 și ε_2, astfel ca $|\overline{f}(x)| \leqslant \varepsilon_1$ sau dacă α se găsește în intervalul

$$\beta \leqslant x \leqslant \gamma, \text{ astfel că } \overline{f}(\beta)\overline{f}(\gamma) \leqslant 0$$

și astfel că

$$\gamma - \beta \leqslant \varepsilon_2$$

Cu aceste două constante pozitive algoritmul de calcul se desfășoară după următoarele etape:

Pentru un a și b dat se testează dacă

$$|\overline{f}(a)| \leqslant \varepsilon_1 \text{ sau } |\overline{f}(b)| \leqslant \varepsilon_1$$

și în acest caz se accepta a sau b ca zerou iar procesul de calcul a luat sfîrșit.

De asemenea se testează dacă

$$\overline{f}(a) \cdot \overline{f}(b) < 0$$

în cazul că această inegalitate nu este satisfăcută, atunci metoda înjumătățirii intervalului nu converge și procesul este oprit, altfel se testează dacă

$$b - a \leqslant \varepsilon_2$$

atunci se calculează $c_0 = \frac{a+b}{2}$ ca un zerou altfel se consideră $a_0 = a$ și $b_0 = b$ și se determină a_1 și b_1 cu ajutorul relațiilor (4).

Diagrama logică de desfășurare al procesului de calcul este dat în fig. 14.2.

În continuare se va scrie un program în BASIC care codifică diagrama logică dată în fig. 14.2 pentru $f(x)=x^3-11$ cu $a=2$ și $b=3$.

```
0001 REM PRØGRAM P2 CAP.14
0005 REM PRØGRAM IN BASIC PENTRU DETERMINAREA RADACINILØR REALE
0006 REM ALE FUNCTIEI F(X)=X↑3-11
0010 REM                METØDA BISECTIEI
0020 READ A,B
```

```
0030 READ E1,E2
0040 DEF FNA(X)=X↑3-11
0050 IF FNA(A)*FNA(B)>0 THEN GØTØ 0160
0060 IF ABS(FNA(A))<=E1 THEN GØTØ 0190
0070 IF ABS(FNA(B))<=E1 THEN GØTØ 0210
0080 IF B-A<=E2 THEN GØTØ 0230
0090 LET C=(A+B)/2
0100 IF FNA(C)=0 THEN GØTØ 0250
0110 IF FNA(A)*FNA(C)<0 THEN GØTØ 0140
0120 LET A=C
0130 GØTØ 0060
0140 LET B=C
0150 GØTØ 0070
0160 PRINT "METØDA BISECTIEI NU CØNVERGE. INTRE LIMITELE INTERVALULUI"
0170 PRINT "NU SE AFLA NICI Ø RADACINA REALA."
0180 STØP
0190 PRINT "ZERØ APRØXIMATIV X=";A
0200 STØP
0210 PRINT "ZERØ APRØXIMATIV X=";B
0220 STØP
0230 PRINT "ZERØ APRØXIMATIV X=";(A+B)/2
0240 STØP
0250 PRINT "ZERØ EXACT X=";C
0260 STØP
0270 DATA 2,3
0280 DATA .000001,.000001
0290 END

* RUN
ZERØ APRØXIMATIV X= 2.22398

STØP AT 0240
*
```

14.2. METODA LUI NEWTON

Această metodă este utilizată la determinarea zerourilor unei funcţii f(x) ea fiind o metodă iterativă. Pentru a pune în evidenţă algoritmul de calcul se pleacă de la teorema lui Taylor.

Dacă α este o soluţie a ecuaţiei f(x)=0, atunci după teorema lui Taylor de dezvoltare în serie se poate scrie

$$f(x)=f(x_n)+(\alpha-x_n)f'(x_n)+\cdots$$

Dar f(α)=0 de unde rezultă

$$\alpha=x_n-\frac{f(x_n)}{f'(x_n)} \quad \text{cu } f'(x_n)\neq 0$$

de unde metoda Newton se defineşte prin

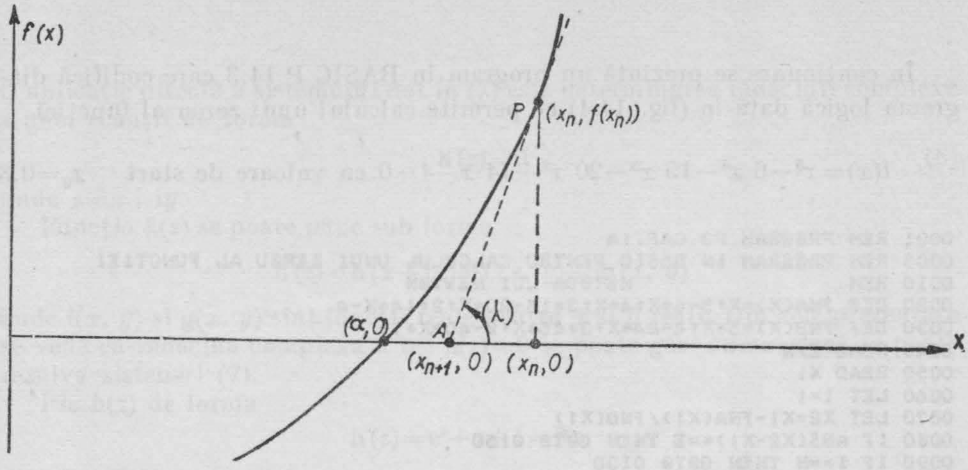
$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}. \tag{5}$$

Această metodă are interpretarea geometrică din fig. 14.3. Metoda Newton este consistentă dacă se defineşte astfel

$$x_{n+1}=\begin{cases} x_n-\dfrac{f(x_n)}{f'(x_n)}, & \text{dacă } f'(x_n)\neq 0 \\ x_n & \text{, dacă } f(x_n)=f'(x_n)=0 \end{cases}$$

Metoda este convergentă dacă f(x) este continuă şi diferenţiabilă $f\in C^2$ în jurul rădăcinii α.

Diagrama logică pentru metoda lui Newton este dată în fig. 14.4.



Interpretarea geometrică a metodei Newton

Fig. 14.3

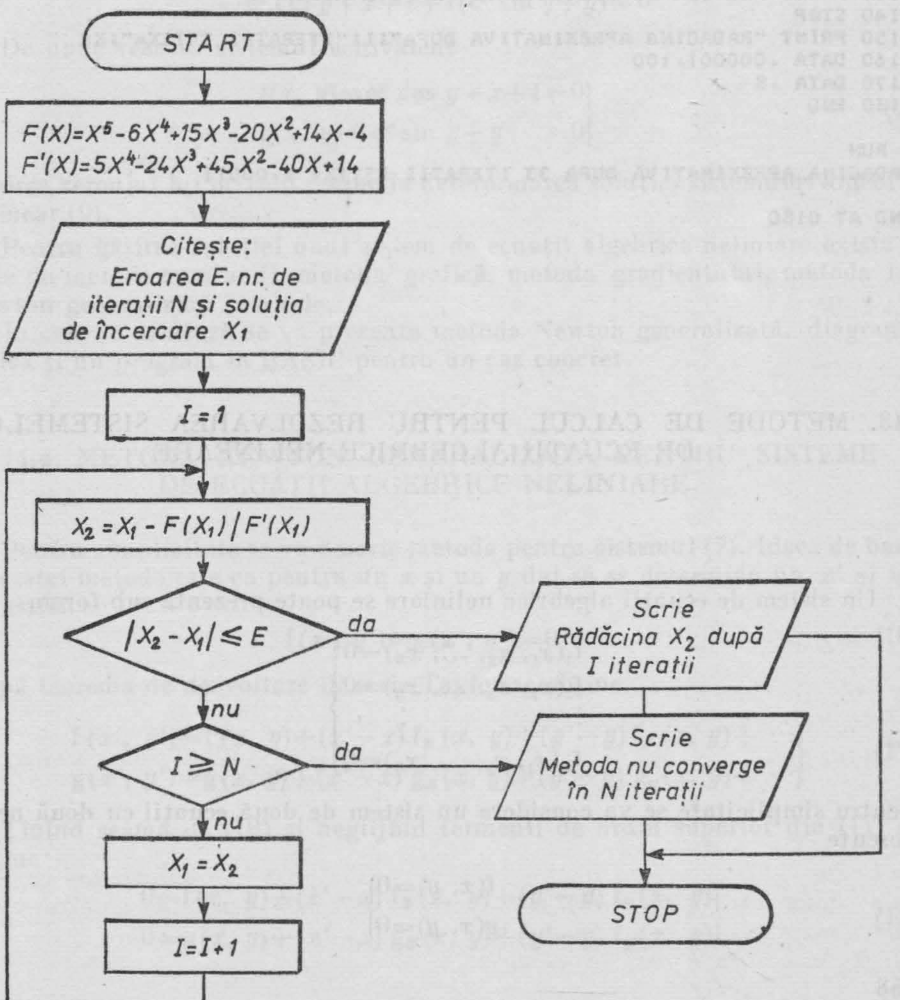


Fig. 14.4

În continuare se prezintă un program în BASIC P 14.3 care codifică diagrama logică dată în (fig. 14.4) ce permite calculul unui zerou al funcţiei

$$f(x)=x^5-6\,x^4-15\,x^3-20\,x^2+14\,x-4=0 \text{ cu valoare de start } x_0=0{,}8$$

```
0001 REM PRØGRAM P3 CAP.14
0003 REM PRØGRAM IN BASIC PENTRU CALCULUL UNUI ZERØU AL FUNCTIEI
0010 REM                  METØDA LUI NEWTØN
0020 DEF FNA(X)=X↑5-6*X↑4+X↑3*15-20*X↑2+14*X-4
0030 DEF FNB(X)=5*X↑4-24*X↑3+45*X↑2-40*X+14
0040 READ E,N
0050 READ X1
0060 LET I=1
0070 LET X2=X1-FNA(X1)/FNB(X1)
0080 IF ABS(X2-X1)<=E THEN GØTØ 0150
0090 IF I>=N THEN GØTØ 0130
0100 LET X1=X2
0110 LET I=I+1
0120 GØTØ 0070
0130 PRINT "METØDA NU CØNVERGE IN";N;"ITERATII."
0140 STØP
0150 PRINT "RADACINA APRØXIMATIVA DUPA";I;"ITERATII ESTEX=";X2
0160 DATA .000001,100
0170 DATA .8
0180 END

* RUN
RADACINA APRØXIMATIVA DUPA 33 ITERATII ESTEX= 1.00011

END AT 0180
*
```

43. METODE DE CALCUL PENTRU REZOLVAREA SISTEMELOR DE ECUAŢII ALGEBRICE NELINEARE

Un sistem de ecuaţii algebrice neliniare se poate prezenta sub forma

$$\left.\begin{aligned} f_1(x_1, x_2, \ldots, x_n)&=0\\ f_2(x_1, x_2, \ldots, x_n)&=0\\ \vdots\qquad\qquad & \ \ \vdots\\ f_n(x_1, x_2, \ldots, x_n)&=0 \end{aligned}\right\} \qquad (6)$$

Pentru simplicitate se va considera un sistem de două ecuaţii cu două necunoscute

$$\left.\begin{aligned} f(x, y)&=0\\ g(x, y)&=0 \end{aligned}\right\} \qquad (7)$$

O aplicaţie directă a sistemului dat în (7) este determinarea rădăcinii complexe a unei ecuaţii de forma

$$h(z)=0 \tag{8}$$

unde $z=x+iy$

Funcţia $h(z)$ se poate pune sub forma

$$h(z)=h(x+iy)=f(x, y)+ig(x, y)$$

unde $f(x, y)$ şi $g(x, y)$ sînt funcţii reale pentru x şi y reali. Din cele prezentate se vede că rădăcina complexă a lui $h(z)=0$ se poate găsi numai dacă se poate rezolva sistemul (7).

Fie $h(z)$ de forma

$$h(z)=e^z+z+1=0$$

Dar

$$h(z)=e^{x+iy}+x+iy+1=e^x e^{iy}+x+iy+1=e^x(\cos y+i\sin y)+x+iy+1=$$
$$=e^x\cos y+x+1+i(e^x\sin y+y)=0$$

De unde rezultă sistemul echivalent

$$\left.\begin{aligned} f(x, y)&=e^x\cos y+x+1=0\\ g(x, y)&=e^x\sin y+y\quad=0\end{aligned}\right\} \tag{9}$$

Găsirea zeroului lui $h(z)=0$ revine la determinarea soluţiei sistemului algebric nelinear (9).

Pentru găsirea soluţiei unui sistem de ecuaţii algebrice neliniare există o serie de metode cum ar fi: metoda grafică, metoda gradientului, metoda lui Newton generalizată şi altele.

În cele ce urmează se va prezenta metoda Newton generalizată, diagrama logică şi un program în BASIC pentru un caz concret.

14.4. METODA NEWTON GENERALIZATĂ PENTRU SISTEME DE ECUAŢII ALGEBRICE NELINIARE

Pentru simplicitate se va descrie metoda pentru sistemul (7). Ideea de bază a acestei metode este ca pentru un x şi un y dat să se determine un x' şi un y' astfel ca

$$f(x', y')=g(x', y')=0 \tag{10}$$

După teorema de dezvoltare în serie Taylor se obţine

$$\left.\begin{aligned} f(x', y')&=f(x, y)+(x'-x)f_x(x, y)+(y'-y)f_y(x, y)+\cdots\\ g(x', y')&=g(x, y)+(x'-x)g_x(x, y)+(y'-y)g_y(x, y)+\cdots\end{aligned}\right\} \tag{11}$$

Ţinînd seama de (10) şi neglijînd termenii de ordin superior din (11) se obţine

$$\left.\begin{aligned} 0&=f(x, y)+(x'-x)f_x(x, y)+(y'-y)f_y(x, y)\\ 0&=g(x, y)+(x'-x)g_x(x, y)+(y'-y)f_y(x, y)\end{aligned}\right\} \tag{12}$$

Dacă se rezolvă sistemul (12) în funcţie de necunoscutele $x'-x$ şi $y'-y$ se obţine

$$\left.\begin{aligned} x'-x&=\frac{-f(x, y)g_y(x, y)+g(x, y)f_y(x, y)}{J(x, y)} \\ y'-y&=\frac{-g(x, y)f_x(x, y)+f(x, y)g_x(x, y)}{J(x, y)} \end{aligned}\right\} \quad (13)$$

unde J este Jacobinul funcţiilor f şi g.

$$J=\det\begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}=f_x g_y-g_x f_y \quad (14)$$

Se observă că dacă $f_y=0$, atunci prima ecuaţie din (13) se reduce la

$$x'-x=-\frac{f(x, y)}{f_n(x, y)}$$

care este metoda lui Newton simplă pentru rezolvarea lui $f(x, y)$ pentru un y fixat.

Dacă se notează $x_0=x$, $y_0=y$, $x_1=x'$ şi $y_1=y'$ atunci (13) devine

$$x_1=x_0+\frac{-f(x_0, y_0)g_y(x_0, y_0)+g(x_0, y_0)f_y(x_0, y_0)}{J(x_0, y_0)}$$

$$y_1=y_0+\frac{-g(x_0, y_0)f_x(x_0, y_0)+f(x_0, y_0)g_x(x_0, y_0)}{J(x_0, y_0)}$$

sau după n iteraţii se obţine

$$x_{n+1}=x_n+\frac{-f(x_n, y_n)g_y(x_n, y_n)+g(x_n, y_n)f_y(x_n, y_n)}{J(x_n, y_n)} \quad (15)$$

$$y_{n+1}=y_n+\frac{-g(x_n, y_n)f_x(x_n, y_n)+f(x_n, y_n)g_x(x_n, y_n)}{J(x_n, y_n)}$$

Diagrama logică şi programul în BASIC sînt date pentru sistemul

$$\begin{bmatrix} f(x, y)=x^3-3xy^2-2x+2=0 \\ g(x, y)=3x^2y-y^3-2y=0 \end{bmatrix} \quad (16)$$

pentru valorile de start $x_0=y_0=1$ folosind relaţiile iterative din (15), unde:

$-f(x_n, y_n)$, $g(x_n, y_n)$ reprezintă valorile funcţiilor f şi g în (x_n, y_n)

$-f_x(x_n, y_n)$, $f_y(x_n, y_n)$, $g_x(x_n, y_n)$, $g_y(x_n, y_n)$ — reprezintă valorile derivatelor în raport cu x şi y în punctul de coordonate (x_n, y_n)

$-J(x_n, y_n)$ reprezintă valoarea jacobianului definit în (14) în punctul de coordonate (x_n, y_n)

În fig. 14.5 este dată diagrama logică a metodei Newton pentru sisteme iar programul P. 14.4 scris în BASIC este folosit pentru calculul soluţiei sistemului neliniar dat prin (16).

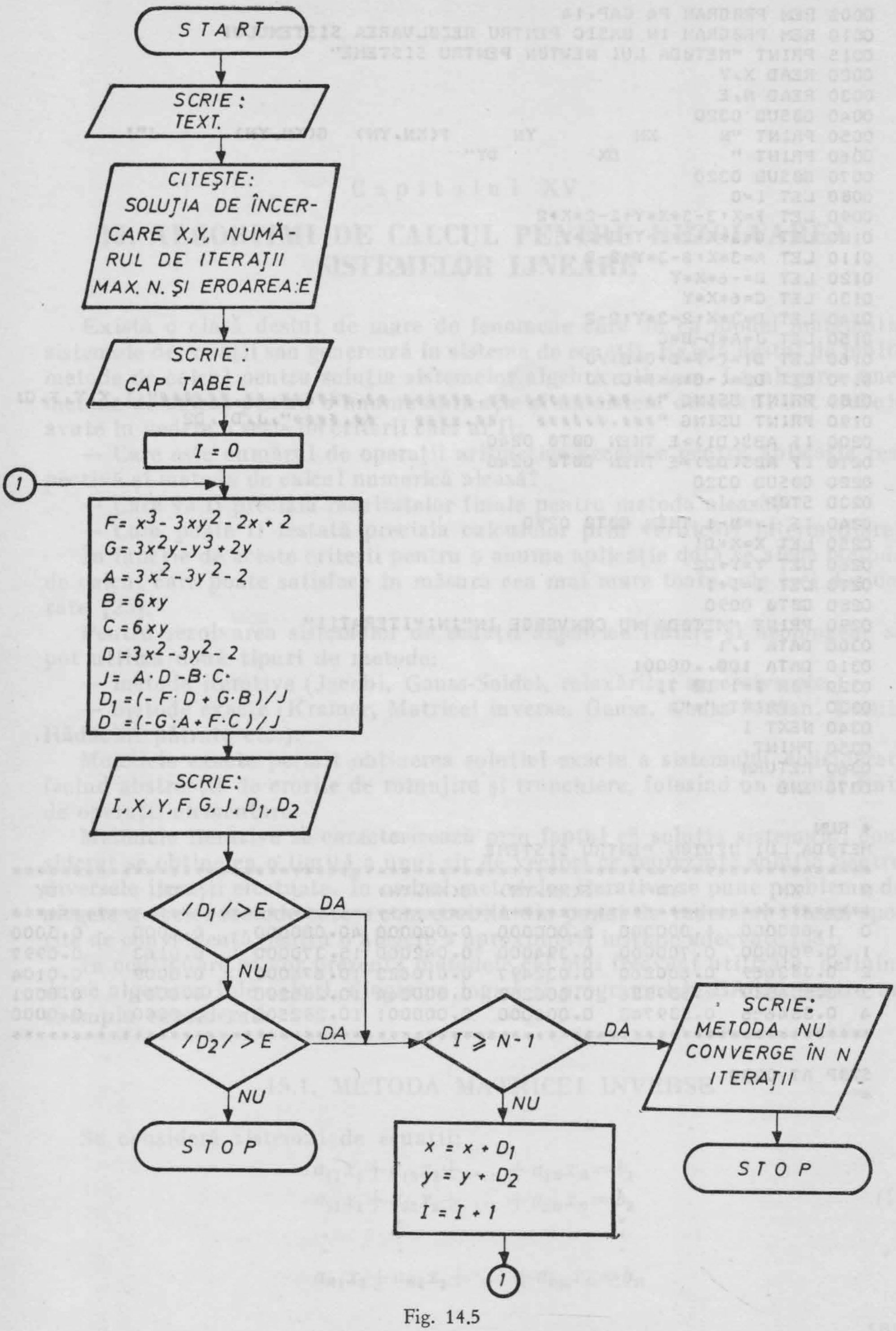

START
SCRIE: TEXT.
CITEŞTE: SOLUŢIA DE ÎNCERCARE X,Y, NUMĂRUL DE ITERAŢII MAX. N. ŞI EROAREA:E
SCRIE: CAP TABEL
I = 0
1
F= x3 - 3xy2 - 2x + 2
G = 3x2y - y3 + 2y
A = 3x2 - 3y2 - 2
B = 6xy
C = 6xy
D = 3x2 - 3y2 - 2
J = A·D - B·C·
D1 = (- F·D + G·B)/J
D = (-G·A + F·C)/J
SCRIE: I, X, Y, F, G, J, D1, D2
/D1/ > E
DA
NU
/D2/ > E
DA
NU
STOP
I ≥ N-1
DA
NU
SCRIE: METODA NU CONVERGE ÎN N ITERAŢII
STOP
x = x + D1
y = y + D2
I = I + 1
1

Fig. 14.5

```
0005 REM PRØGRAM P4 CAP.14
0010 REM PRØGRAM IN BASIC PENTRU REZØLVAREA SISTEMULUI
0015 PRINT "METØDA LUI NEWTØN PENTRU SISTEME"
0020 READ X,Y
0030 READ N,E
0040 GØSUB 0320
0050 PRINT "N      XN           YN         F(XN,YN)  G(XN,YN)          J";
0060 PRINT "          DX           DY"
0070 GØSUB 0320
0080 LET I=0
0090 LET F=X↑3-3*X*Y↑2-2*X+2
0100 LET G=3*X↑2*Y-Y↑3-2*Y
0110 LET A=3*X↑2-3*Y↑2-2
0120 LET B=-6*X*Y
0130 LET C=6*X*Y
0140 LET D=3*X↑2-3*Y↑2-2
0150 LET J=A*D-B*C
0160 LET D1=(-F*D+G*B)/J
0170 LET D2=(-G*A+F*C)/J
0180 PRINT USING "# ##.###### ##.###### ##.###### ##.######",I,X,Y,F,G;
0190 PRINT USING "###.######    ##.####    ##.####",J,D1,D2
0200 IF ABS(D1)>E THEN GØTØ 0240
0210 IF ABS(D2)>E THEN GØTØ 0240
0220 GØSUB 0320
0230 STØP
0240 IF I>=N-1 THEN GØTØ 0290
0250 LET X=X+D1
0260 LET Y=Y+D2
0270 LET I=I+1
0280 GØTØ 0090
0290 PRINT "METØDA NU CØNVERGE IN";N;"ITERATII"
0300 DATA 1,1
0310 DATA 100,.00001
0320 FØR I=1 TØ 71
0330   PRINT "*";
0340 NEXT I
0350 PRINT
0360 RETURN
0370 END

* RUN
METØDA LUI NEWTØN PENTRU SISTEME
***********************************************************************
N    XN         YN        F(XN,YN)  G(XN,YN)      J             DX         DY
***********************************************************************
0  1.000000  1.000000  2.000000  0.000000 40.000000      0.1000     0.3000
1  0.900000  0.700000  0.394000  0.042000 15.370000      0.0163     0.0997
2  0.883669  0.600260  0.032497  0.010623 10.674000      0.0009     0.0104
3  0.884539  0.589836  0.000252  0.000244 10.285200      0.0001     0.0001
4  0.884646  0.589743  0.000000  0.000001 10.282500      0.0000     0.0000
***********************************************************************

STØP AT 0230
*
```

Capitolul XV

15. ALGORITMI DE CALCUL PENTRU REZOLVAREA SISTEMELOR LINEARE

Există o clasă destul de mare de fenomene care au ca model matematic sistemele de ecuaţii sau generează în sisteme de ecuaţii. Există destul de multe metode de calcul pentru soluţia sistemelor algebrice liniare. La alegerea unei metode de calcul pentru o anume aplicaţie şi un sistem de calcul dat trebuie avute în vedere o serie de criterii cum ar fi:

— Care este numărul de operaţii aritmetice necesare pentru aplicaţia respectivă şi metoda de calcul numerică aleasă?

— Care va fi precizia rezultatelor finale pentru metoda aleasă?

— Cum poate fi testată precizia calculelor prin verificări intermediare?

În funcţie de aceste criterii pentru o anume aplicaţie dată se alege metoda de calcul care poate satisface în măsura cea mai mare toate cele trei deziderate [25].

Pentru rezolvarea sistemelor de ecuaţii algebrice liniare şi neomogene se pot utiliza două tipuri de metode:

— metode iterative (Jacobi, Gauss-Seidel, relaxărilor succesive etc.)

— metode exacte (Kramer, Matricei inverse, Gauss, Gauss-Jordan, Grout, Rădăcini pătrate etc.).

Metodele exacte permit obţinerea soluţiei exacte a sistemului considerat, făcînd abstracţie de erorile de rotunjire şi trunchiere, folosind un număr finit de operaţii elementare.

Metodele iterative se caracterizează prin faptul că soluţia sistemului considerat se obţine ca o limită a unui şir de vectori ce reprezintă soluţia pentru diversele iteraţii efectuate. În cadrul metodelor iterative se pune problema de alegere a acelei metode care e convenabilă din punct de vedere al vitezei sporite de convergenţă pentru o alegere a aproximării iniţiale adecvate [58].

În continuare se vor prezenta metodele cel mai frecvent utilizate, definindu-se algoritmul de calcul, diagrama logică şi programul în BASIC pentru un exemplu considerat.

15.1. METODA MATRICEI INVERSE

Se consideră sistemul de ecuaţii:

$$\begin{matrix} a_{11}x_1+a_{12}x_2+\ \dots\ +a_{1n}x_n=b_1 \\ a_{21}x_1+a_{22}x_2+\ \dots\ +a_{2n}x_n=b_2 \\ \vdots \qquad \vdots \qquad\qquad \vdots \\ a_{n1}x_1+a_{n2}x_2+\ \dots\ +a_{nn}x_n=b_n \end{matrix} \tag{1}$$

sau sub formă matricială (1) se poate scrie astfel

$$Ax=b \tag{2}$$

unde

$$A\equiv(a_{ij})=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}; \; x=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}; \; b=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3}$$

Sistemul (2), are o soluţie unică dacă şi numai dacă det $A \neq 0$, adică A este nesingulară (există A^{-1}).

$$A^{-1}A = AA^{-1} = I(\delta_{ij}) = \begin{cases} 0 \text{ dacă } i \neq j \\ 1 \text{ dacă } i=j \end{cases} \quad i, j=1, \dots, n \tag{4}$$

Considerînd că A^{-1} există şi are coeficienţii de forma

$$A^{-1}\equiv(c_{ij}),\ i, j=1, 2, \dots, n, \tag{5}$$

multiplicînd (2) cu A^{-1} la stînga şi ţinînd seama că

$Ix=x$, atunci se obţine:

$$x=A^{-1}b \tag{6'}$$

sau sub formă dezvoltată

$$x_i=\sum_{j=1}^{n} c_{ij}b_j,\quad i=1, 2, \dots, n \tag{6}$$

Relaţia (6) permite calculul tuturor componentelor vectorului necunoscut $x^T(x_1, x_2, \dots, x_n)$.

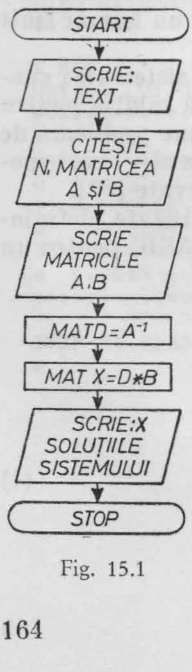


Fig. 15.1

Dacă se consideră elementele c_{ij} ale matricei A^{-1} cunoscute se observă din (6) că pentru fiecare componentă $x_i \in x$ sînt necesare n operaţii de înmulţire şi $(n-1)$ operaţii de adunare.

Considerînd (+, —, /, *) ca operaţii elementare, atunci pentru calculul unei componente a vectorului x sînt necesare $n+(n-1)$ operaţii elementare, deci pentru calculul vectorului x sînt necesare $n^2+n\,(n-1)$ operaţii elementare, din care n^2 operaţii de înmulţire şi $n\,(n-1)$ operaţii de adunare.

Exemplu

Se consideră sistemul

$$\begin{aligned} 5\,x_1+7\,x_2+6\,x_3+5\,x_4+6\,x_5&=29 \\ 7\,x_1+10\,x_2+8\,x_3+7\,x_4+5\,x_5&=37 \\ 6\,x_1+8\,x_2+10\,x_3+9\,x_4+6\,x_5&=39 \\ 5\,x_1+7\,x_2+9\,x_3+10\,x_4+8\,x_5&=39 \\ 8\,x_1+7\,x_2+6\,x_3+9\,x_4+10\,x_5&=40 \end{aligned}$$

Soluţia exactă a acestui sistem este vectorul

$$x^T = [1, 1, 1, 1, 1,]$$

În continuare se prezintă programul în BASIC şi soluţia calculată, prin metoda matricei inverse folosind algoritmul dat în (6′) şi (6).

```
0001 REM PRØGRAM PI CAP. 15.
0002 PRINT "PRØGRAM BASIC PENTRU REZØLVAREA SISTEMELØR PRIN METØDA"
0003 PRINT "MATRICII INVERSE FØLØSIND ALGØRITMUL DAT IN (6') SI (6)""
0020 PRINT "INTRØDUCETI DIMENSIUNEA SISTEMULUI"
0030 INPUT N
0035 DIM A[N,N],B[N,1],X[N,1],D[N,N]
0050 FØR I=1 TØ N
0055   FØR J=1 TØ N
0060     READ A(I,J)
0065   NEXT J
0070 NEXT I
0075 PRINT "MATRICEA CØEFICIENTILØR ESTE"
0080 MAT PRINT A
0090 FØR I=1 TØ N
0100   READ B[I,1]
0110 NEXT I
0113 PRINT "TERMENII LIBERI SINT"
0115 MAT PRINT B
0120 MAT D=INV(A)
0130 MAT X=D*B
0140 PRINT "SØLUTIA SISTEMULUI ESTE"
0150 FØR I=1 TØ N
0160 V PRINT "X("I")="X[I,1]
0170 NEXT I
0175 DATA 5,7,6,5,6,7,10,8,7,5,6,8,10,9,6,5,7,9,10,8,8,7,6,9,10
0177 DATA 29,37,39,39,40
0180 END

* RUN
PRØGRAM BASIC PENTRU REZØLVAREA SISTEMELØR PRIN METØDA
MATRICII INVERSE FØLØSIND ALGØRITMUL DAT IN (6') SI (6)
INTRØDUCETI DIMENSIUNEA SISTEMULUI
 ? 5
MATRICEA CØEFICIENTILØR ESTE

 5          7          6          5          6
 7          10         8          7          5
 6          8          10         9          6
 5          7          9          10         8
 8          7          6          9          10
TERMENII LIBERI SINT

 29
 37
 39
 39
 40
SØLUTIA SISTEMULUI ESTE

X( 1)= .999874
X( 2)= 1.00103
X( 3)= .999993
X( 4)= .999956
X( 5)= .999998

END AT 180
*
```

15.2. METODA GAUSS DE ELIMINARE

Această metodă are o mare aplicabilitate la rezolvarea sistemelor algebrice liniare şi pentru calculele necesare inversării matricilor. Trebuie menţionat faptul că ordinea ecuaţiilor într-un sistem şi poziţia necunoscutelor nu este unică, datorită acestui fapt ecuaţiile pot fi schimbate între ele şi necunoscutele renumerotate (lucru de care trebuie ţinut cont la interpretarea rezultatelor) în scopul unei mai bune condiţionări a sistemului, operaţii ce sînt impuse de precizia rezultatelor. La baza metodei Gauss de eliminare stă următorul procedeu:

• prima ecuaţie a sistemului este folosită la eliminarea lui x_1 din următoarele $n-1$ ecuaţii.

Dacă s-a făcut operaţia de condiţionare a sistemului atunci $a_{11}\neq 0$, se calculează $n-1$ constante de multiplicare

$$m_i=\frac{a_{i1}}{a_{11}},\quad i=2, 3, \ldots, n \tag{7}$$

Se înmulţeşte prima ecuaţie cu m_i şi se scade din ecuaţia i pentru $i=2, 3, \ldots, n$. În urma acestui proces se obţin următorii coeficienţi pentru sistemul (1)

$$a_{ij}^{(1)}=a_{ij}-m_i\, a_{1j} \qquad i=2, \ldots, n \tag{8}$$

$$b_i^{(1)}=b_i-m_i b_1 \qquad j=1, 2, \ldots, n$$

Din (7) şi prima relaţie din (8) se vede că

$$a_{i1}^{(1)}=0,\quad i=2, \ldots, n \tag{9}$$

Relaţia (9) scoate în evidenţă că necunoscuta x_1 a fost eliminată din ultimele $n-1$ ecuaţii ale sistemului original.

• a doua ecuaţie transformată în prima etapă este utilizată la eliminarea lui x_2 din următoarele $n-2$ ecuaţii ş.a.m.d.

La etapa k se elimină x_k prin definirea constantelor de multiplicare

$$m_i^{(k-1)}=\frac{a_{ij}^{(k-1)}}{a_{kk}^{(k-1)}},\quad i=k+1, k+2, \ldots, n \tag{10}$$

unde $a_{kk}^{(k-1)}\neq 0$, (în urma operaţiei de condiţionare) şi

$$a_{ij}^{(k)}=a_{ij}^{(k-1)}-m_i^{(k-1)}\, a_{kj}^{(k-1)} \qquad j=k, \ldots, n \tag{11}$$

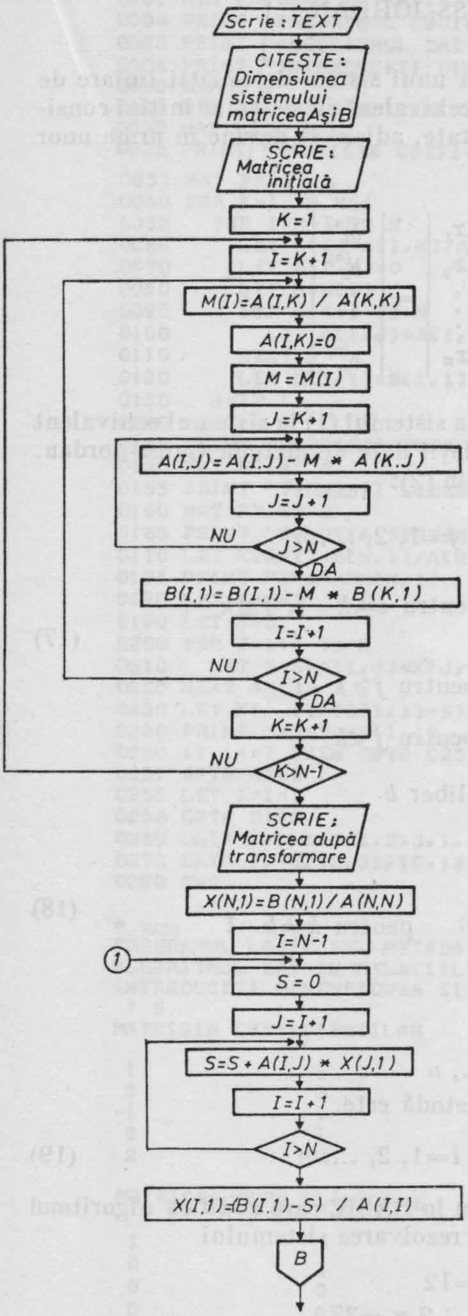
şi

$$b_i^{(k)}=b_i^{(k-1)}-m_i^{(k-1)}\, b_k^{(k-1)} \qquad i=k+1, \ldots, n \tag{12}$$

iar k ia valorile consecutive de la 1, 2, ..., $n-1$. În etapa în care $k=n-1$ are loc eliminarea lui x_{n-1} din ultima ecuaţie a sistemului, obţinîndu-se sistemul (13).

În urma acestui proces format din $n-1$ eliminăm sistemul (1) devine

$$\left.\begin{aligned}
a_{11}x_1+a_{12}x_2+\ldots+a_{1k}\,x_k+\ldots+a_{1n}x_n&=b_1\\
a_{21}^{(1)}\,x_2+\ldots+a_{2k}^{(1)}\,x_k+\ldots+a_{2n}^{(1)}x_n&=b_2^{(1)}\\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots&\\
a_{kk}^{(k-1)}x_k+\ldots+a_{kn}^{(k-1)}x_n&=b_k^{(k-1)}\\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots&\\
a_{nn}^{(n-1)}\,x_n&=b_n^{(n-1)}
\end{aligned}\right\} \tag{13}$$



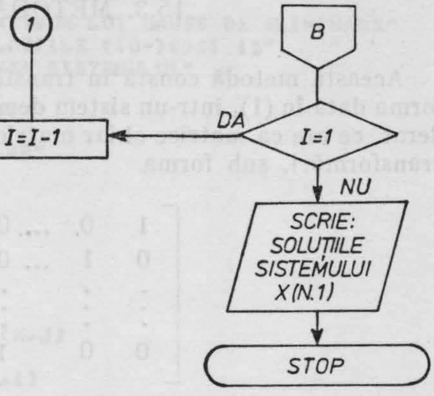


Fig. 15 2

Sistemul (13) obținut din (1) are o formă superior triunghiulară și este mult mai ușor de rezolvat, el poate fi scris sub formă matricială astfel

$$Cx = D \tag{14}$$

Sistemele de ecuații (2) și (14) sînt echivalente dacă admit aceeași soluție într-o anumită vecinătate din spațiul soluției ($x_1, x_2, \dots, x_n$) unde matricile C și D sînt obținute din A și B prin diverse transformări elementare în urma procesului de $n-1$ eliminări.

Soluția sistemului obținut în (13) se calculează cu ajutorul următoarelor relații:

$$x_n = b_n^{(n-1)} / a_{nn}^{(n-1)}$$

$$x_{n-1} = (b_{n-1}^{(n-2)} - a_{n-1,\,n}^{(n-2)}\, x_n)/a_{n-1,\,n-1}^{(n-2)}$$

$$\dots\dots\dots\dots\dots\dots\dots \tag{15}$$

$$x_j = (b_j^{(j-1)} - a_{jn}^{(j-1)}\, x_n - \dots$$

$$\dots - a_{j,\,j+1}^{(j-1)}\, x_{j+1})/a_{jj}^{(j-1)},$$

$$j = n-2, \dots, 3, 2, 1$$

Diagrama logică după care se execută algoritmul dat în (10)—(12) și (15) este dată în fig. 15.2.

15.3. METODA GAUSS-JORDAN

Această metodă constă în transformarea unui sistem de ecuaţii liniare de forma dată în (1), într-un sistem de ecuaţii echivalent cu sistemul iniţial considerat, ce are ca matrice chiar matricea unitate, adică (1) devine în urma unor transformări, sub forma

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(n)} \\ b_2^{(n)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

Algoritmul de calcul prin care se ajunge de la sistemul (1) la sistemul echivalent (16) se poate prezenta prin următoarele relaţii date de metoda Gauss-Jordan, relaţii de transformare asupra matricei A din (2):

$$\begin{cases} a_{ij}^{(1)} = a_{ij} & i, j=1, 2, \dots, n \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - \dfrac{a_{i,\,k-1}^{(k-1)}}{a_{k-1,\,k-1}^{(k-1)}} a_{k-1,j}^{(k-1)} & \text{pentru } i \neq k-1,\ j \geqslant k-1 \\ a_{k-1,\,j}^{(k)} = a_{k-1,\,j}^{(k-1)} & \text{pentru } j \geqslant k-1 \\ a_{i,j}^{(k)} = a_{ij}^{(k-1)} & \text{pentru } j < k-1 \end{cases} \quad (17)$$

şi asupra coeficienţilor vectorului termen liber b

$$\begin{cases} b_i^{(k)} = b_i \\ b_i^{(k)} = b_i^{(k-1)} - \dfrac{a_{i,\,k-1}^{(k-1)}}{a_{k-1,\,k-1}^{(k-1)}} b_{k-1}^{(k-1)} & \text{pentru } k \neq k-1 \\ b_{k-1}^{(k)} = b_{k-1}^{(k-1)} \end{cases} \quad (18)$$

Pentru relaţiile din (17) şi (18) $k=2, \dots, n$

Soluţia sistemului (16) prin această metodă este

$$x_i = b_i^{(n)} \qquad \text{pentru } i=1, 2, \dots, n \quad (19)$$

În continuare se va prezenta un program în BASIC care codifică algoritmul dat în relaţiile (10—12) şi (15), utilizat la rezolvarea sistemului

$$\begin{aligned} x_1+x_2+x_3+x_4+x_5&=12 \\ 2\,x_1+3\,x_2+x_3+5\,x_4+2\,x_5&=35 \\ -x_1+x_2-5\,x_3+3\,x_4+6\,x_5&=10 \\ 3\,x_1+x_2+7\,x_3-2\,x_4-3\,x_5&=12 \\ 2\,x_1+2\,x_2+2\,x_3+x_4-3\,x_5&=10 \end{aligned}$$

```
0001 REM PRØGRAM P2 CAP.15
0004 PRINT "PRØGRAMUL LØGIC PT. METØDA LUI GAUSS DE ELIMINARE"
0005 PRINT "ALGØRITMUL DAT IN RELATIILE (10-12)SI 15"
0006 PRINT "INTRØDUCETI DIMENSIUNEA SISTEMULUI"
0010 INPUT N
0020 DIM A[N,N],B[N,1],X[N,1],M[N]
0030 MAT READ A,B
0036 PRINT "MATRICEA CØEFICIENTILØR"
0037 MAT PRINT A
0040 FØR K=1 TØ N-1
0050    FØR I=K+1 TØ N
0060      LET M[I]=A[I,K]/A[K,K]
0070      LET A[I,K]=0
0080      LET M=M[I]
0090      FØR J=K+1 TØ N
0100        LET A[I,J]=A[I,J]-M*A[K,J]
0110      NEXT J
0120      LET B[I,1]=B[I,1]-M*B[K,1]
0130    NEXT I
0140 NEXT K
0145 PRINT "MATRICEA DUPA ELIMINARE"
0150 MAT PRINT A
0155 PRINT "TERMENII LIBERI SINT"
0160 MAT PRINT B
0165 PRINT "SØLUTIA SISTEMULUI ESTE"
0170 LET X[N,1]=B[N,1]/A[N,N]
0175 PRINT "X"N"="X[N,1]
0180 LET I=N-1
0190 LET S=0
0200 FØR J=I+1 TØ N
0210    LET S=S+A[I,J]*X[J,1]
0220 NEXT J
0230 LET X[I,1]=(B[I,1]-S)/A[I,I]
0240 PRINT "X"I"="X[I,1]
0250 IF I<>1 THEN GØTØ 0255
0251 GØTØ 0260
0255 LET I=I-1
0256 GØTØ 0190
0260 DATA 1,1,1,1,1,2,3,1,5,2,-1,1,-5,3,6,3,1,7,-2,-3,2,2,2
0270 DATA 1,-3,12,35,10,12,10
0280 END

* RUN
PRØGRAMUL LØGIC PT. METØDA LUI GAUSS DE ELIMINARE
ALGØRITMUL DAT IN RELATIILE (10-12)SI 15
INTRØDUCETI DIMENSIUNEA SISTEMULUI
 ? 5
MATRICEA CØEFICIENTILØR

 1          1          1          1          1
 2          3          1          5          2
-1          1         -5          3          6
 3          1          7         -2         -3
 2          2          2          1         -3

MATRICEA DUPA ELIMINARE

 1          1          1          1          1
 0          1         -1          3          0
 0          0         -2         -2          7
 0          0          0         -1          1
 0          0          0          0         -6
TERMENII LIBERI SINT

 12
 11
 0
-2
-12
```

```
SØLUTIA SISTEMULUI ESTE
X 5= 2
X 4= 4
X 3= 3
X 2= 2
X 1= 1

END AT 0280
*
```

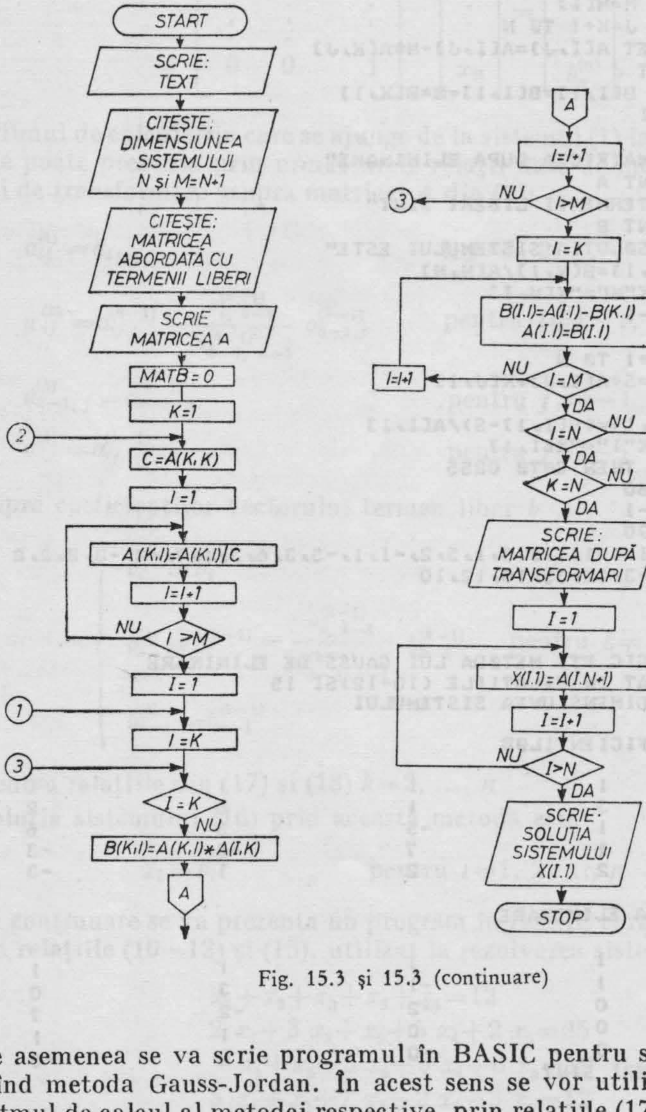


Fig. 15.3 şi 15.3 (continuare)

De asemenea se va scrie programul în BASIC pentru sistemul de mai sus utilizînd metoda Gauss-Jordan. În acest sens se vor utiliza relaţiile date în algoritmul de calcul al metodei respective, prin relaţiile (17), (18) şi (19), după care se vor compara cele două soluţii găsite.

```
0005 REM PRØGRAM P3 CAP.15
0010 PRINT "PRØGRAMUL BASIC PT. METØDA GAUSS-JØRDAN"
0015 PRINT "FØLØSIND ALGØRITMUL DAT PRIN RELATIILE (17),(18),(19)"
0030 READ N,M
0040 DIM A[N,N+1],B[N,N+1],X[N,1]
0041 DIM X[N,1]
0050 MAT READ A
0052 PRINT "MATICEA CØEFICIENTILØR BØRDATA CU TERMENII LIBERI"
0053 MAT PRINT A;
0055 MAT B=ZER
0060 FØR K=1 TØ N
0070   LET C=A[K,K]
0080    FØR J=1 TØ M
0090     LET A[K,J]=A[K,J]/C
0100   NEXT J
0110    FØR I=1 TØ N
0120     LET J=K
0130     IF I=K THEN GØTØ 0210
0135     LET B[K,J]=A[K,J]*A[I,K]
0150     LET J=J+1
0160     IF J>M THEN GØTØ 0180
0170     GØTØ 0130
0180     FØR J=K TØ M
0185       LET B[I,J]=A[I,J]-B[K,J]
0195       LET A[I,J]=B[I,J]
0200     NEXT J
0210   NEXT I
0220 NEXT K
0221 PRINT "MATRICEA DUPA TRANSFØRMARI BØRDATA CU TERMENII LIBERI"
0230 MAT PRINT A;
0235 PRINT "SØLUTIA SISTEMULUI ESTE"
0236 FØR I=1 TØ N
0237   LET X[I,1]=A[I,N+1]
0238   PRINT "X("I")="X[I,1]
0239 NEXT I
0240 DATA 5,6,1,1,1,1,1,12,2,3,1,5,2,35,-1,1,-5,3,6,10,3,1,7,-2
0241 DATA -3,12,2,2,2,1,-3,10
0250 END

* RUN
PRØGRAMUL BASIC PT. METØDA GAUSS-JØRDAN
FØLØSIND ALGØRITMUL DAT PRIN RELATIILE (17),(18),(19)
MATICEA CØEFICIENTILØR BØRDATA CU TERMENII LIBERI

 1  1  1  1  1  12
 2  3  1  5  2  35
-1  1 -5  3  6  10
 3  1  7 -2 -3  12
 2  2  2  1 -3  10
MATRICEA DUPA TRANSFØRMARI BØRDATA CU TERMENII LIBERI

 1  0  0  0  0  1
 0  1  0  0  0  2
 0  0  1  0  0  3
 0  0  0  1  0  4
 0  0  0  0  1  2
SØLUTIA SISTEMULUI ESTE
X( 1)= 1
X( 2)= 2
X( 3)= 3
X( 4)= 4
X( 5)= 2
```

În paragrafele precedente au fost prezentate cîteva metode directe pentru rezolvarea sistemelor liniare de ordinul n, metode care în general necesită $n^3/3$ operaţii elementare. Trebuie menţionat că în calculele efectuate cu aceste metode, eroare care se introduce prin rotunjire devine destul de mare şi este dependentă de n care reprezintă numărul ecuaţiilor sistemului.

Metodele iterative constau în considerarea unui vector iniţial $\mathbf{x}^{(0)}$ ca vector soluţie şi cu ajutorul unui algoritm de calcul iterativ se determină un şir de aproximaţii succesive pentru vectorul soluţie $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)} \ldots$ care în principiu trebuie să tindă către soluţia exactă a sistemului. În cazul în care metoda converge destul de rapid, procedeul ia sfîrşit foarte repede şi se obţine o aproximaţie foarte bună.

Fie sistemul (2) şi $\mathbf{x}^{(0)} \in \mathrm{R}^{(n)}$ un vector oarecare cu care se construieşte un şir de vectori $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$ cu ajutorul unei formule de recurenţă de forma

$$\mathbf{x}^{(k+1)} = F_k(\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}) \tag{20}$$

unde F_k este o funcţie vectorială (nu neapărat liniară) de argumentele sale. În general F_k depinde de matricea A a sistemului. Relativ la funcţia F_k se pot face următoarele afirmaţii:

— Dacă F_k este liniară pentru orice k, procesul de aproximaţii succesive este liniar.

— Dacă F_k depinde de un singur vector

$$\mathbf{x}^{k+1} = \mathrm{F}_k(\mathbf{x}^{(k)}) \tag{21}$$

atunci procesul de aproximaţii este de ordinul întîi.

Dacă se consideră că matricea A din sistemul (2) este nesingulară, adică există A^{-1} atunci ea se poate descompune într-un infinit număr de moduri sub forma

$$A = D + C$$

unde D şi C sînt matrici de acelaşi ordin cu A, iar sistemul

$$A\mathbf{x} = b \text{ se poate scrie sub forma } D\mathbf{x} + C\mathbf{x} = b$$

sau

$$D\mathbf{x} = \mathbf{b} - C\mathbf{x} \tag{22}$$

Dacă se consideră un vector $\mathbf{x}^{(0)}$, se poate cu ajutorul lui (22) construi un şir de vectori $\mathbf{x}^k$, astfel

$$D\mathbf{x}^{(k)} = \mathbf{b} - C\mathbf{x}^{(k-1)}, \quad k = 1, 2, \ldots \tag{23}$$

Matricea D se alege aşa fel ca să existe D^{-1}, şi atunci (23) devine

$$\mathbf{x}^{(k)} = D^{-1}\mathbf{b} + D^{-1}C\mathbf{x}^{(k-1)} \quad k = 1, 2, \ldots \tag{24}$$

Pentru analiza convergentei şirului de vectori soluţie obţinuţi prin iteraţii $\mathbf{x}^{(k)}$, către vectorul soluţie $\mathbf{x}$, se impune introducerea unei matrici

$$P = -D^{-1}C$$

şi un vector de eroare

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}, \quad k = 0, 1, 2, \ldots$$

Făcînd diferenţa între (23) şi (22) rezultă

$$D(\mathbf{x}^{(k)}-\mathbf{x})=C(\mathbf{x}-\mathbf{x}^{(k-1)})$$

sau

$$D\mathbf{e}^{(k)}=-C\mathbf{e}^{(k-1)} \quad \text{sau } \mathbf{e}^{(k)}=-D^{-1}C\mathbf{e}^{(k-1)}),$$

de unde rezultă că

$$\mathbf{e}^{(k)}=-D^{-1}C\mathbf{e}^{(k-1)}=P\mathbf{e}^{(k-1)}=P^2\mathbf{e}^{(k-2)}=\ldots=P^{(k)}\mathbf{e}^{(0)},\ k=1,\ 2,\ \ldots \tag{25}$$

Din relaţia (25) se vede că o condiţie necesară pentru convergenţa şirului $\{\mathbf{x}^{(k)}\}\rightarrow\mathbf{x}$, este ca

$$\lim_{k\to\infty}\mathbf{e}^{(k)}=0, \text{ adică } \lim_{k\to\infty}\mathrm{P}^{(k)}=0 \tag{26}$$

această condiţie fiind şi necesară dacă metoda e convergentă pentru orice $\mathbf{e}^{(0)}$.

A doua limită din (26) are sens numai dacă matricea P este convergentă, matricea P este convergentă dacă şi numai dacă valorile proprii ale lui P, adică raza spectrală este subunitară ($\rho(P)<1$, unde $\rho(P)\equiv\max|\lambda_i|$, unde λ_i sînt valorile proprii ale matricii P).

După această prezentare generală se va trece la analiza cîtorva din metodele iterative cele mai utilizate şi eficiente din punct de vedere al numărului de operaţii, necesarul de memorie şi controlul erorilor.

15.5. METODA ITERATIVĂ JACOBI

Această metodă se mai numeşte şi metoda iteraţiilor simultane. Dacă matricea A din (2) este nesingulară şi elementele a_{ii} de pe diagonala principală sînt toate diferite de zero, atunci A se poate scrie sub forma

$$A=D+T+S \tag{27}$$

unde

— A este o matrice pătratică

— D matrice diagonală $a_{ii}\neq 0$ $i=1,\ 2,\ \ldots,\ n$

— T matrice strict inferior triunghiulară ($a_{ij}=0,\ i\leqslant j$ şi $a_{ij}\neq 0,\ i>j$)

— S matrice strict superior triunghiulară ($a_{ij}\neq 0,\ i\leqslant j$ şi $a_i=0\ i\geqslant j$)

Ţinînd seamă de expresia lui A din (27) sistemul (2) devine

$$(D+T+S)\mathbf{x}=\mathbf{b}$$

sau

$$Dx=\mathbf{b}-(T+S)\mathbf{x} \tag{28}$$

Relaţia (28) se poate scrie dezvoltat sub forma

$$a_{ii}x_i=b_i-\sum_{\substack{j=1\\ j\neq i}}^{n}a_{ij}x_j \tag{29}$$

Trecîndu-se la un proces iterativ se obţine

$$a_{ii}x_i^{(k)}=b_i-\sum_{\substack{j=1\\ j\neq i}}^{n}a_{ij}x_j^{(k-1)} \qquad i=1,\ 2,\ \ldots,\ n \tag{30}$$

unde $x_i^{(0)}$ $i=1,\ 2,\ \ldots,\ n$ sînt estimaţi iniţial pentru componentele vectorului soluţie $\mathbf{x}$.

Relaţia (30) se mai poate scrie sub forma

$$x_i^{(k)} = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k-1)} \qquad i=1, 2, \dots, n \tag{31}$$

sau matriceal sub forma

$$\mathbf{x}^{(k)} = D^{-1}\mathbf{b} - D^{-1}(T+S)\mathbf{x}^{(k-1)}, \; k \geqslant 0 \tag{32}$$

Matricea $-D^{-1}(T+S)$ se mai numeşte şi matricea Jacobi asociată matricei A, dacă această matrice se notează cu $P=-D^{-1}(T+S)$ atunci schema (32) devine

$$\mathbf{x}^{(k)} = D^{-1}\mathbf{b} + P\mathbf{x}^{(k-1} \tag{33}$$

iar relaţia (28) se poate scrie sub forma

$$\mathbf{x} = D^{-1}\mathbf{b} + P\mathbf{x} \tag{34}$$

Pentru analiza convergenţei şirului de vectori $x^{(k)}$ către soluţia exactă x a sistemului (2), este necesară analiza vectorului eroare care apare în metoda lui Jacobi

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}, \; k=0, 1, 2, \dots \tag{35}$$

Dacă se face diferenţa între (30) şi (34) rezultă

$$\mathbf{x}^{(k)} - \mathbf{x} = P(\mathbf{x}^{(k-1)} - \mathbf{x}), \; k > 0, \tag{36}$$

de unde rezultă vectorul eroare în cazul metodei iterative Jacobi:

$$\mathbf{e}^{k} = P\mathbf{e}^{(k-1)} = P^2\mathbf{e}^{(k-2)} = \dots = P^{(k)}\mathbf{e}^{(0)}, \; k=0, 1, 2, \dots \tag{37}$$

15.6. METODA ITERATIVĂ GAUSS-SEIDEL

Această metodă mai este întîlnită în literatură şi sub denumirea de metoda iteraţiilor succesive.

Metoda iterativă Jacobi calculează componentele vectorului $\mathbf{x}^{(k+1)}$ folosind în exclusivitate componentele vectorului $\mathbf{x}^{(k)}$ de la iteraţia k

Metoda Gauss-Seidel e o modificare a metodei Jacobi, deoarece la calculul componentelor $x_i^{(k+1)}$ se folosesc toate componentele vectorului $x^{(k+1)}$ care sînt cunoscute pînă în acel moment (adică $x_1^{(k+1)}, x_2^{(k+1)} \dots, x_{i-1}^{(k+1)}$) şi componentele vectorului $\mathbf{x}^{k}$ (adică $x_{i+1}^{(k)}, x_{i+2}^{(k)}, \dots, x_n^{(k)}$). De unde se vede că, componentele vectorului $\mathbf{x}^{(k)}$ sînt utilizate succesiv în ordinea în care ele au fost obţinute adică sistemul (1) se pune sub forma

$$a_{ii}x_i = b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j \qquad i=1, 2, \dots, n \tag{38}$$

Relaţia (38) se poate folosi într-un proces iterativ astfel

$$a_{ii}\mathbf{x}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}\mathbf{x}^{(k)} - \sum_{j=i+1}^{n} a_{ij}\mathbf{x}^{(k-1)} \qquad \begin{matrix} i=1, 2, \dots, n \\ k=1, 2, \dots \end{matrix} \tag{39}$$

Relaţia (39) se mai poate scrie sub forma

$$a_{ii}x^{(k)}+\sum_{j=1}^{i-1}a_{ij}x^{(k)}=b_i-\sum_{j=i+1}^{n}a_{ij}x^{(k-1)} \tag{40}$$

sau matricial astfel

$$(D+T)\mathbf{x}^{(k)}=\mathbf{b}-S\mathbf{x}^{(k-1)},\quad k\geqslant 1 \tag{41}$$

În cazul în care există $(D+T)^{-1}$, atunci (41) se poate pune sub forma

$$\mathbf{x}^{(k)}=(D+T)^{-1}\mathbf{b}-(D+T)^{-1}S\mathbf{x}^{(k-1)},\quad k\geqslant 1 \tag{42}$$

Dacă se notează $P=-(D+T)^{-1}S$, atunci matricea P se numeşte „matricea Gauss-Seidel" asociată matricei A.

Pentru a pune în evidenţă elementele privind convergenţa metodei Gauss-Seidel se scrie sistemul (39) şi sistemul (38) sub forma:

$$x_i^{(k)}=\frac{b_i}{a_{ii}}-\sum_{j=1}^{i-1}\frac{a_{ij}}{a_{ii}}x_j^{(k)}-\sum_{j=i+1}^{n}\frac{a_{ij}}{a_{ii}}x_j \qquad \begin{matrix} i=1,2,\ldots,n\\ k=1,2,\ldots\end{matrix} \tag{43}$$

$$x_i=+\frac{b_i}{a_{ii}}-\sum_{j=1}^{i-1}\frac{a_{ij}}{a_{ii}}x_j-\sum_{j=i+1}^{n}\frac{a_{ij}}{a_{ii}}x_j$$

Făcînd diferenţa celor două relaţii din (43) rezultă:

$$x_i^{(k)}-x_i=-\sum_{j=1}^{i-1}\frac{a_{ij}}{a_{ii}}(x_j^{(k)}-x_j)-\sum_{j=i+1}^{n}\frac{a_{ij}}{a_{ii}}(x_j^{(k-1)}-x_j)$$

Introducîndu-se eroarea sub forma

$$e_i^{(k)}=x_i^{(k)}-x_i \qquad \begin{matrix} k=0,1,2,\ldots\\ i=1,2,\ldots,n\end{matrix}$$

relaţia (44) devine

$$e_i^{(k)}=-\sum_{j=1}^{i-1}\frac{a_{ij}}{a_{ii}}e^{(k)}-\sum_{j=i+1}^{n}\frac{a_{ij}}{a_{ii}}e^{(k-1)} \qquad \begin{matrix} i=1,2,\ldots,n\\ k=1,2,\ldots\end{matrix} \tag{45}$$

Acest test privind convergenţa metodei Gauss-Seidel este uşor de făcut cu ajutorul relaţiei (45).

Din algoritmul dat prin (42) sau prima relaţie din (43) se vede că procesul de convergenţă este mult mai rapid faţă de procesul iterativ Jacobi. Necesarul de memorie este mai mic în metoda Gauss-Seidel decît în metoda Jacobi, fiind necesar un singur spaţiu de memorie pentru vectorul soluţie, spaţiu care va conţine la iteraţia k, componentele $x_j^{(k)}$ ce au fost calculate şi componentele $x_i^{(k-1)}$ pentru $i>j$ şi $i=j, j+1, \ldots, n$

```
0005 REM PRØGRAM P4 CAP.15.
0010 PRINT "PRØGRAM IN BASIC PT. METØDA GAUSS-SEIDEL"
0020 PRINT "DATA PRIN ALGØRITMUL (42) RESPECTIV (43)"
0030 INPUT N,E1,K1
0031 DIM A[N,N]
0040 DIM D[N,N],T[N,N],S[N,N],X[N,1],Y[N,1],B[N,1],P[N,N],R[N,N]

0041 MAT D=ZER
0042 MAT S=ZER
0043 MAT T=ZER
0045 MAT READ A,B,X
0050 DIM Z[N,N],Q[N,N],H[N,1],G[N,1],E[N,1]
0051 FØR I=1 TØ N
0052   FØR K=I+1 TØ N
0053     IF K>N THEN GØTØ 0060
0054     LET S[I,K]=A[I,K]
0055     LET T[K,I]=A[K,I]
0056     LET D[I,I]=A[I,I]
0057   NEXT K
0058 NEXT I
0060 LET D[I,I]=A[I,I]
0063 PRINT "MATRICEA D ESTE"
0064 MAT PRINT D;
0065 PRINT "MATRICEA T ESTE"
0066 MAT PRINT T;
0067 PRINT "MATRICEA S ESTE"
0068 MAT PRINT S;
0069 LET K=0
0070 MAT Z=D+T
0080 MAT R=INV(Z)
0082 MAT Q=R*S
0083 FØR I=1 TØ N
0084   FØR J=1 TØ N
0085     LET L[I,J]=-Q[I,J]
0090   NEXT J
0100 NEXT I
0110 MAT H=Q*X
0120 MAT G=R*B
0130 MAT Y=G+H
0150 LET M=0
0160 FØR I=1 TØ N
0170   LET M=M+ABS(Y[I,1]-X[I,1])
0180 NEXT I
0190 PRINT "VECTØRUL ERØARE ARE VALØAREA GLØBALA="M
0200 FØR I=1 TØ N
0210   IF ABS(Y[I,1]-X[I,1])>E1 THEN GØTØ 0300
0220 NEXT I
0230 LET K=K+1
0240 PRINT "DUPA "K"ITERATII SØLUTIA CU ERØARE<"E1"ESTE"
0250 FØR I=I TØ N
0260   PRINT "X"I"="Y[I,1]
0270 NEXT I
0280 DATA 100,7,6,5,4,3,2,1,7,234,6,5,4,3,2,1,6,6,546,5,4,3,2,1
0281 DATA 5,5,5,345,4,3,2,1,4,4,4,4,564,3,2,1,3,3,3,3,3,654,2,1
0282 DATA 2,2,2,2,2,2,652,1,1,1,1,1,1,1,1,1000
0283 DATA -2,-3,45,34,98,2,3,12
0284 DATA 0,0,0,0,0,0,0,0
0290 END

0300 LET K=K+1
0310 IF K>K1 THEN GØTØ 0340
0320 MAT X=Y
0330 GØTØ 0070
0340 PRINT "SØLUTIA NU A FØST GASITA DUPA"K1"ITERATII"
0350 END
* RUN
PRØGRAM IN BASIC PT. METØDA GAUSS-SEIDEL
DATA PRIN ALGØRITMUL (42) RESPECTIV (43)
 ? 8 ? .00001 ? 100
```

```
MATRICEA D ESTE

100  0   0   0   0   0   0   0
0   234  0   0   0   0   0   0
0    0  546  0   0   0   0   0
0    0   0  345  0   0   0   0
0    0   0   0  564  0   0   0
0    0   0   0   0  654  0   0
0    0   0   0   0   0  652  0
0    0   0   0   0   0   0  1000
MATRICEA T ESTE

0  0  0  0  0  0  0  0
7  0  0  0  0  0  0  0
6  6  0  0  0  0  0  0
5  5  5  0  0  0  0  0
4  4  4  4  0  0  0  0
3  3  3  3  3  0  0  0
2  2  2  2  2  2  0  0
1  1  1  1  1  1  1  0
MATRICEA S ESTE

0  7  6  5  4  3  2  1
0  0  6  5  4  3  2  1
0  0  0  5  4  3  2  1
0  0  0  0  4  3  2  1
0  0  0  0  0  3  2  1
0  0  0  0  0  0  2  1
0  0  0  0  0  0  0  1
0  0  0  0  0  0  0  0
VECTØRUL ERØARE ARE VALØAREA GLØBALA= .402389
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 2.69203E-02
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 7.61268E-04
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 4.5714E-06
DUPA  4ITERATII SØLUTIA CU ERØARE< .00001ESTE
X 1=-3.18262E-03
X 2=-5.37754E-03
X 3= 8.47296E-02
X 4= 9.95147E-02
X 5= .172554

X 6= 1.48933E-03
X 7= 3.5463E-03
X 8= 1.16467E-02

END AT 0290
*
```

Exemplu. Se consideră sistemul

$$\begin{bmatrix} 100 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 7 & 234 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 6 & 546 & 5 & 4 & 3 & 2 & 1 \\ 5 & 5 & 5 & 345 & 4 & 3 & 2 & 1 \\ 4 & 4 & 4 & 4 & 564 & 3 & 2 & 1 \\ 3 & 3 & 3 & 3 & 3 & 654 & 2 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 652 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1000 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} -2 \\ -3 \\ -45 \\ 34 \\ 98 \\ 2 \\ 3 \\ 12 \end{bmatrix}$$

Soluţia obţinută prin metoda Jacobi este dată de programul P 15.5 iar soluţia obţinută prin metoda Gauss-Seidel este dată de programul P 15.4.

```
0005 REM PRØGRAM P5 CAP.15.
0010 PRINT "PRØGRAM IN BASIC PT. METØDA JACØBI"
0020 PRINT "DATA PRIN ALGØRITMUL (31) RESPECTIV (33)"
0030 INPUT N,E1,K1
0031 DIM A[N,N]
0040 DIM D[N,N],T[N,N],S[N,N],X[N,1],Y[N,1],B[N,1],P[N,N],R[N,N]
0041 MAT D=ZER
0042 MAT S=ZER
0043 MAT T=ZER
0045 MAT READ A,B,X
0050 DIM Z[N,N],Q[N,N],H[N,1],G[N,1],E[N,1]
0051 FØR I=1 TØ N
0052   FØR K=I+1 TØ N
0053     IF K>N THEN GØTØ 0060
0054     LET S[I,K]=A[I,K]
0055     LET T[K,I]=A[K,I]
0056     LET D[I,I]=A[I,I]
0057   NEXT K
0058 NEXT I
0060 LET D[I,I]=A[I,I]
0063 PRINT "MATRICEA D ESTE"
0064 MAT PRINT D;
0065 PRINT "MATRICEA T ESTE"
0066 MAT PRINT T;
0067 PRINT "MATRICEA S ESTE"
0068 MAT PRINT S;
0069 LET K=0
0070 MAT Z=INV(D)
0080 FØR I=1 TØ N
0082   FØR J=1 TØ N
0083     LET Q[I,J]=-Z[I,J]
0084   NEXT J
0085 NEXT I
0090 MAT R=T+S
0100 MAT P=Q*R
0110 MAT H=Z*B
0120 MAT G=P*X
0130 MAT Y=H+G
0150 LET M=0
0160 FØR I=1 TØ N
0170   LET M=M+ABS(Y[I,1]-X[I,1])
0180 NEXT I
0190 PRINT "VECTØRUL ERØARE ARE VALØAREA GLØBALA="M
0200 FØR I=1 TØ N
0210   IF ABS(Y[I,1]-X[I,1])>E1 THEN GØTØ 0300
0220 NEXT I
0230 LET K=K+1
0240 PRINT "DUPA "K"ITERATII SØLUTIA CU ERØARE<"E1"ESTE"
0250 FØR I=1 TØ N
0260   PRINT "X"I"="Y[I,1]
0270 NEXT I
0280 DATA 100,7,6,5,4,3,2,1,7,234,6,5,4,3,2,1,6,6,546,5,4,3,2,1
0281 DATA 5,5,5,345,4,3,2,1,4,4,4,4,564,3,2,1,3,3,3,3,3,654,2,1
0282 DATA 2,2,2,2,2,2,652,1,1,1,1,1,1,1,1,1000
0283 DATA -2,-3,45,34,98,2,3,12
0284 DATA 0,0,0,0,0,0,0,0
0290 END
0300 LET K=K+1
0310 IF K>K1 THEN GØTØ 0340
0320 MAT X=Y
0330 GØTØ 0090
0340 PRINT "SØLUTIA NU A FØST GASITA DUPA"K1"ITERATII"
```

```
0350 END

* RUN
PRØGRAM IN BASIC PT. METØDA JACØBI
DATA PRIN ALGØRITMUL (31) RESPECTIV (33)
 ? 8 ? .00001 ? 100
MATRICEA D ESTE

100  0   0   0   0   0   0   0
0   234  0   0   0   0   0   0
0   0   546  0   0   0   0   0
0   0   0   345  0   0   0   0
0   0   0   0   564  0   0   0
0   0   0   0   0   654  0   0
0   0   0   0   0   0   652  0
0   0   0   0   0   0   0   1000
MATRICEA T ESTE

0   0   0   0   0   0   0   0
7   0   0   0   0   0   0   0
6   6   0   0   0   0   0   0
5   5   5   0   0   0   0   0
4   4   4   4   0   0   0   0
3   3   3   3   3   0   0   0
2   2   2   2   2   2   0   0
1   1   1   1   1   1   1   0
MATRICEA S ESTE

0   7   6   5   4   3   2   1
0   0   6   5   4   3   2   1
0   0   0   5   4   3   2   1
0   0   0   0   4   3   2   1
0   0   0   0   0   3   2   1
0   0   0   0   0   0   2   1
0   0   0   0   0   0   0   1
0   0   0   0   0   0   0   0
VECTØRUL ERØARE ARE VALØAREA GLØBALA= .407207
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 3.16003E-02
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 2.63841E-03
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 2.3501E-04
VECTØRUL ERØARE ARE VALØAREA GLØBALA= 2.02146E-05
DUPA  5ITERATII SØLUTIA CU ERØARE< .00001ESTE
X 1=-3.54822E-02
X 2=-.018942
X 3= 8.08255E-02
X 4= 9.60945E-02
X 5= .172848
X 6= 1.67428E-03
X 7= 3.67224E-03
X 8= 1.16993E-02

END AT 0290
*
```

Rezultatele obţinute prin metoda Jacobi (Programul P 15.5) şi metoda Gauss-Seidel (Programul P 15.4) au fost verificate prin metoda matricei inverse (Programul P 15.6) de unde se vede că diferenţa între soluţiile obţinute prin metodele iterative şi metoda exactă este de ordinul 10^{-7}.

De asemenea se poate vedea din programele P 15.4 şi P 15.5 că metoda Gauss-Seidel converge mult mai rapid decît metoda Jacobi, cum de altfel era şi de aşteptat.

```
0001 REM PRØGRAM P6 CAP.15.
0002 REM PRØGRAM IN BASIC PENTRU  METØDA MATRICEI INVERSE
0005 PRINT "ALGØRITMUL DAT IN (6')SI (6)"

0020 PRINT "INTRØDUCETI DIMENSIUNEA SISTEMULUI "
0030 INPUT N
 035 DIM A[N,N],B[N,1],X[N,1],D[N,N]
0050 FØR I=1 TØ N
0055   FØR J=1 TØ N
0060     READ A[I,J]
0065   NEXT J
0070 NEXT I
0075 PRINT "MATRICEA CØEFICIENTILØR ESTE"

0080 MAT PRINT A;
0090 FØR I=1 TØ N
0100   READ B[I,1]
0110 NEXT I
0113 PRINT "TERMENII LIBERI SINT"
0115 MAT PRINT B;
0120 MAT D=INV(A)
0130 MAT X=D*B
0140 PRINT "SØLUTIA SISTEMULUI ESTE"
0150 FØR I=1 TØ N
0160    PRINT "X("I")="X[I,1]
0170 NEXT I
0175 DATA 100,7,6,5,4,3,2,1,7,234,6,5,4,3,2,1,6,6,546,5,4,3,2,1

0176 DATA 5,5,5,345,4,3,2,1,4,4,4,4,564,3,2,1,3,3,3,3,3,654,2,1

0177 DATA 2,2,2,2,2,2,652,1,1,1,1,1,1,1,1000
0178 DATA -2,-3,45,34,93,2,3,12
0180 END

* RUN
ALGØRITMUL DAT IN (6')SI (6)
INTRØDUCETI DIMENSIUNEA SISTEMULUI
 ? 8
MATRICEA CØEFICIENTILØR ESTE

 100  7   6   5   4   3   2   1
 7   234  6   5   4   3   2   1
 6   6   546  5   4   3   2   1
 5   5   5   345  4   3   2   1
 4   4   4   4   564  3   2   1
 3   3   3   3   3   654  2   1
 2   2   2   2   2   2   652  1
 1   1   1   1   1   1   1   1000
TERMENII LIBERI SINT

-2
-3
 45
 34
 98
 2
 3
 12

SØLUTIA SISTEMULUI ESTE
X( 1)=-3.54828E-02
X( 2)=-1.89423E-02
X( 3)= 8.08253E-02
X( 4)= 9.60944E-02
X( 5)= .172847
X( 6)= 1.67421E-03
X( 7)= 3.67219E-03
X( 8)= 1.16993E-02

END AT 0180
*
```

Capitolul XVI

16. ALGORITMI PENTRU REZOLVAREA ECUAŢIILOR DIFERENŢIALE ORDINARE ŞI A SISTEMELOR DE ECUAŢII DIFERENŢIALE ORDINARE

O ecuaţie diferenţială de ordinul întîi

$$\frac{dy}{dx}=f(x, y) \text{ sau } F(x, y, y')=0$$

poate fi întîlnită ca model matematic pentru o varietate de fenomene fizice sau sociale, în cadrul unui număr mare de discipline.

Exemple de astfel de fenomene se pot da: circuite electrice, probleme de mecanică, viteza de multiplicare a bacteriilor, viteza de dezintegrare a materiilor radioactive (fizică atomică), viteza de creştere a populaţiei (statistică), fenomene de creştere economică etc. Ecuaţiile diferenţiale ordinare pot fi întîlnite sub forma:

— ecuaţii diferenţiale ordinare de ordinul întîi

$$y'=\frac{dy}{dx}=f(x, y)$$

— ecuaţii diferenţiale de ordinul $r\geqslant 2$

$$y^{(r)}=\frac{d^r y}{dx^r}=f(x, y, y', \ldots, y^{(r-2)}, y^{(r-1)})$$

— sistem de n ecuaţii diferenţiale ordinare de ordinul întîi

$$y_i'=f_i(x, y_1, y_2, \ldots, y_n),\ i=1, 2, \ldots, n$$

Pentru majoritatea problemelor care apar în practică şi implică ecuaţii diferenţiale ordinare există formulate şi o serie de condiţii iniţiale sau la limită, impuse de necesitatea fenomenului respectiv, condiţii care fac ca soluţia să fie unică.

Fie ecuaţia diferenţială ordinară

$$y'=f(x, y) \tag{1}$$

cu condiţia iniţială

$$y(x_0)=y_0 \tag{2}$$

Problema găsirii funcţiei $y(x)$, care să verifice identic ecuaţia (1) cu condiţia (2) poartă numele de problema Cauchy sau problema cu condiţii iniţiale. În cadrul metodelor numerice pe care le vom prezenta, facem presupunerea că

s-a făcut un studiu teoretic al problemei enunțate, deci sînt satisfăcute condițiile teoremei de existență și unicitate, în cadrul metodelor respective urmărindu-se calcularea efectivă a soluției.

Pentru foarte multe ecuații diferențiale ordinare, nu se pot aplica metodele clasice de integrare întîlnite, sau aceste metode conduc la calcule extrem de laborioase.

De aceea sîntem nevoiți să apelăm la așa zisele metode „aproximative“ care sînt de două tipuri:

1. metode analitice, care dau aproximarea soluției sub forma unor expresii analitice,

2. metode numerice, în cadrul cărora soluția se obține sub forma unui șir de valori, plecînd de la o valoare inițială a soluției.

Dintre metodele analitice amintim: metoda aproximațiilor succesive și metoda determinării puterii sub forma unei serii de puteri.

Rezolvarea numerică a problemei cu condiții inițiale dată prin (1) și (2) constă în determinarea unui număr de puncte $y_1, y_2, \ldots, y_n$ care aproximează valorile exacte $y(x_1), y(x_2), \ldots, y(x_n)$ ale curbei integrale $y=y(x)$, curbă care trece prin punctul inițial (x_0, y_0), cu pasul de integrare

$$h=x_{i+1}-x_i, \quad i=1, 2, \ldots, n$$

Metodele numerice pentru integrarea ecuațiilor diferențiale ordinare la rîndul lor, se pot împărți în două categorii:

— Metode cu pași separați, care necesită pentru calculul ordonatei y_{i+1} cunoașterea coordonatelor punctului anterior (x_i, y_i) și mărimea pasului de integrare. Foarte multe din aceste metode apelează la dezvoltarea în serie Taylor a funcției $y=y(x)$ în jurul unui punct:

$$y(x_{i+1})=y(x_i)+hy'(x_i)+\frac{h}{2}y''(x_i)+\ldots$$

— Metode cu pași legați, care pentru calculul ordonatei y_{i+1}, necesită cunoașterea pasului de integrare h și cunoașterea mai multor puncte anterioare $(x_i, y_i); (x_{i-1}, y_{i-1}), \ldots$

Șirul de valori $y_i, y_{i-1}, \ldots, y_2, y_1$ se determină în general folosind metode numerice cu pași separați.

Pentru metodele cu pași legați se utilizează egalitatea evidentă.

$$y(x_{i+1})-y(x_{i-j})=\int_{x_{i-j}}^{x_{i+1}} y'(x)\mathrm{d}x$$

unde integrantul $y'=\mathrm{f}(x, y)$ se aproximează cu un polinom de interpolare.

Ambele tipuri de metode cu pași separați sau cu pași legați, se folosesc pentru generarea șirului de valori aproximative $y_1, y_2, \ldots, y_n$ sub forma unui algoritm explicit sau implicit.

Pentru o metodă cu pași separați, un algoritm de tip explicit este de forma:

$$y_{i+1}=y_i+h\varphi(x_i, y_i, h) \quad i=0, 1, 2, \ldots, n$$

iar un algoritm implicit este de forma:

$$y_{i+1}^{C} = y_i+h\varphi\ (x_i, y_i, x_{i+1}, y_{i+1}^{P}, h) \quad i=0, 1, 2, \ldots,$$

Aproximaţia y_{i+1}^{P} care apare în partea dreaptă a egalităţii poartă numele de valoare prescrisă (sau cu predicţie) şi se calculează în general cu un algoritm explicit, iar aproximaţia y_{i+1}^{C} poartă numele de valoare corectată a ordonatei y_{i+1}.

Într-o manieră asemănătoare se construiesc algoritmi expliciţi şi impliciţi pentru metodele numerice cu paşi legaţi.

16.1. METODA DEZVOLTĂRII ÎN SERIE TAYLOR

Fie ecuaţia diferenţială

$$y'=f(x, y) \text{ cu condiţia iniţială } y(x_0)=y_0 \tag{3}$$

Se pune problema găsirii unei valori aproximative y_1 pentru valoarea exactă $y(x_1)$ de pe curba integrală $y=y(x)$. Dacă soluţia analitică $y(x)$ este dezvoltabilă în serie Taylor în jurul lui x_0, atunci:

$$y(x)=y(x_0)+\frac{x-x_0}{1\,!}y'(x_0)+\frac{(x-x_0)^2}{2\,!}y''(x_0)+\dots+\frac{(x-x_0)^P}{p\,!}y^{(P)}(x_0)+$$
$$+\frac{(x-x_0)^{P+1}}{(p+1)\,!}y^{P+1}(x_0+\theta x), 0<\theta<1 \tag{4}$$

sau pentru $x=x_1$ şi $h=x_1-x_0$; relaţia (4) devine

$$y(x_1)=y(x_0)+\frac{h}{1\,!}y'(x_0)+\frac{h^2}{2}y''(x_0)+\dots+\frac{h^P}{p'}y^P(x_0)+$$
$$+\frac{h^{P+1}}{(p+1)\,!}y^{P+1}(x_0+\theta x_1) \tag{5}$$

Din relaţia (5) se vede că pentru determinarea valorii $y(x_1)$ este necesară determinarea valorilor $y(x_0)$, $y'(x_0)$, ..., $y^p(x_0)$, h fiind cunoscut.

În acest sens avem:

$y(x_0)=y_0$ din condiţia iniţială dată în (3)

$y'(x_0)=f(x_0, y(x_0))=f(x_0, y_0)$ se poate calcula din ecuaţia diferenţială dată în (3), iar

$$y''(x)=\frac{dy'(x)}{dx}=\frac{d}{dx}f[x, y(x)]=\frac{\partial f}{\partial x}+\frac{\partial f}{\partial y}y'=\frac{\partial f}{\partial x}+f\frac{\partial f}{\partial y}$$

de unde rezultă

$$y''(x_0)=\frac{\partial f(x_0, y_0)}{\partial x}+f(x_0, y_0)\frac{\partial f(x_0, y_0)}{\partial y}$$

Dacă se ţine seamă că f depinde de x direct cît şi prin intermediul lui y, atunci $y''(x_0)$ are expresia

$$y''(x_0)=\left\{\frac{\partial^2 f}{\partial x^2}+2f\frac{\partial^2 f}{\partial x\,\partial y}+f^2\frac{\partial^2 f}{\partial y^2}+\frac{\partial f}{\partial y}\left[\frac{\partial f}{\partial x}+f\frac{\partial f}{\partial y}\right]\right\}\Bigg|_{(x_0,\, y_0)}$$

Din evaluarea lui $y''(x_0)$ se vede cum calculele se complică din ce în ce mai mult din cauza acestor derivări, motiv pentru care se reţine din dezvoltarea (4) numai un număr mic de termeni, din această cauză nu se va găsi valoarea

exactă $y(x_1)$ dată de formula (5), dar se va găsi o valoare aproximativă y_1 care aproximează pe $y(x_1)$ cu o precizie destul de bună.

O dată ce aproximaţia y_1 a fost determinată se poate calcula aproximaţia y_2 prin înlocuirea lui x_1 cu x_2 şi x_0 cu x_1 în relaţia (5).

În final se va obţine un şir de valori ale soluţiei $y(x)$, $[y_0, y_1, y_2, ..., y_n]$ care vor fi toate aproximative afară de y_0. Datorită calculelor laborioase pe care le implică derivările, această metodă devine foarte incomodă pentru un p destul de mare.

Exemplul 1

Se consideră ecuaţia diferenţială ordinară

$$y'=x^2+y^2, \quad y(0)=1 \tag{6}$$

Prin derivare succesivă se obţine

$$\begin{aligned}
&y'=x^2+y^2 && y'(0)=x_0^2+y_0^2=1\\
&y''=2x+2yy' && y''(0)=2x_0+2y_0y_0'=2\\
&y'''=2+2yy''+2y'^2 && y'''(0)=2+2y_0y''+2y_0'^2=8\\
&y^{(4)}=2yy'''+6y'y'' && y_0^{(4)}=2y_0y_0'''+6y_0y_0''=28
\end{aligned} \tag{7}$$

Dacă în relaţia (4) $p=4$ se obţine

$$y(x)=y(x_0)+\frac{x-x_0}{1!}y'(x_0)+\frac{(x-x_0)^2}{2!}y''(x_0)+\frac{(x-x_0)^3}{3!}y'''(x_0)+$$
$$+\frac{(x-x_0)}{4!}y^{(4)}(x_0)$$

Pentru $x_0=0$ şi substituind expresiile derivatelor de mai sus calculate în punctul $x=0$, această ultimă dezvoltare Taylor trunchiată după derivata de ordinul patru devine

$$y(x)=1+x+2\frac{x^2}{2!}+8\frac{x^3}{3!}+28\frac{x^4}{4!}=1+x+x^2+\frac{4}{3}x^3+\frac{7}{6}x^4 \tag{8}$$

În continuare se prezintă programul în BASIC pentru determinarea soluţiei numerice a ecuaţiei diferenţiale ordinare date în (6), folosindu-se expresia Taylor (8), pentru intervalul $x\in[0, 2]$ cu un pas $h=0,2$.

```
0001 REM PRØGRAM P1 CAP.16.
0010 PRINT "PRØGRAMUL BASIC PT. DET. SØL. NUMERICE A EC."
0015 PRINT "          DIFERENTIALE ØRDINARE"
0020 PRINT "DATA IN (6) FØLØSINDUSE EXPRESIA TAYLØR (8)"
0030 PRINT "PT. INTERVALUL 0<=X<=2 CU PASUL H=0.2"
0035 INPUT P2,B6,H1
0040 READ X,H,X1
0041 LET N=B6/H+1
0042 DIM V[N,1]
```

```
0050 PRINT "ECUATIA DIFERENTIALA ESTE Y'=X↑2+Y↑2"
0053 LET I=1
0055 PRINT "CU CØNDITIA INITIALA Y(0)=1"
0060 LET Y=1+X+X↑2+4/3*X↑3+7/6*X↑4
0062 IF I>N THEN GØTØ 0070
0063 LET V[I,1]=Y
0064 LET I=I+1
0070 PRINT "PT. X="X,"Y="Y
0080 LET X=X+H
0090 IF X<=2 THEN GØTØ 0060
0100 DATA 0,.2,2
0590 FØR I=1 TØ N
0600   IF I<>1 THEN GØTØ 0680
0610   FØR J=H1 TØ 71
0612     IF J=H1 THEN PRINT  TAB(J)"0";
0615     IF J=71 THEN GØTØ 0665
0620     LET A=V[I,1]/P2+H1
0630     IF J=A THEN GØTØ 0660
0640     PRINT  TAB(J)"-";
0650     GØTØ 0670
0660     PRINT  TAB(J)"*";
0662     GØTØ 0670
0665     PRINT  TAB(J)"Y"
0670   NEXT J
0680   LET S=V[I,1]/P2
0690   LET S=S+H1
0700   PRINT  TAB(H1)"!"; TAB(S)"*"
0710 NEXT I
0720 FØR I=1 TØ 6
0730   IF I=6 THEN GØTØ 0760
0740   PRINT  TAB(H1)"!"
0750   GØTØ 0770
0760   PRINT  TAB(H1)"X"
0770 NEXT I
0780 END
*

* RUN
PRØGRAMUL BASIC PT. DET. SØL. NUMERICE A EC.
        DIFERENTIALE ØRDINARE
DATA IN (6) FØLØSINDUSE EXPRESIA TAYLØR (8)
PT. INTERVALUL 0<=X<=2 CU PASUL H=0.2

 ? .2 ? 1 ? 5
ECUATIA DIFERENTIALA ESTE Y'=X↑2+Y↑2
CU CØNDITIA INITIALA Y(0)=1
PT. X= 0     Y= 1
PT. X= .2    Y= 1.25253
PT. X= .4    Y= 1.6752
PT. X= .6    Y= 2.3992
PT. X= .8    Y= 3.60053
PT. X= 1     Y= 5.5
PT. X= 1.2   Y= 8.3632
PT. X= 1.4   Y= 12.5005
PT. X= 1.6   Y= 18.2672
PT. X= 1.8   Y= 26.0632
PT. X= 2     Y= 36.3333
```

```
0----*--------------------------------------------------------------------- Y
!     *
!      *
!         *
!           *
!               *
!                    *
!
!
!
!
X
END AT 0780
*
```

16.2. METODA COEFICIENŢILOR NEDETERMINAŢI

Această metodă este uneori mai simplă decît metoda Taylor. Se scrie dezvoltarea în serie Taylor sub forma

$$y(x)=a_0+a_1(x-x_0)+a_2(x-x_0)^2+\dots \tag{9}$$

Se doreşte evaluarea coeficienţilor a_k, nu prin calcularea lui $y^k(x_0)$, ci prin substituirea expresiei (9) în ecuaţia diferenţială şi egalarea coeficienţilor puterilor $(x-x_0)$ de acelaşi ordin.

Dacă se consideră ecuaţia diferenţială ordinară (6), pentru $x_0=0$, (9) devine după trunchiere de forma

$$y=a_0+a_1x+a_2x^2+a_3x^2+a_3x^3+a_4x^4+a_5x^5 \tag{10}$$

$$y'=a_1+2\,a_2x+3\,a_3x^2+4\,a_4x^3+5\,a_5x^4$$

Dacă se înlocuiesc expresiile y şi y' în (6) rezultă

$$a_1+2\,a_2x+3\,a_3x^2+4\,a_4x^3+5\,a_5x^4=x^2+(a_0+a_1x+a_2x^2+a_3x^3+a_4x^4+a_5x^5)^2,$$

după ordonare se obţine

$$a_1+2\,a_2x+3\,a_3x^2+4\,a_4x^3+5\,a_5x^4=a^2+2\,a_0a_1x+(1+2\,a_0a_2+a_1)x^2+$$
$$+(2\,a_0a_3+2\,a_1a_2)x^3+(2\,a_0a_4+2\,a_1a_3+a^2)x^4+\dots$$

În urma identificării coeficienţilor lui x se obţine sistemul

$$a_1=a_0^2,\ 2\,a_2=2\,a_0a_1;\ 3\,a_3=1+2\,a_0a_2+a^2,\ 4\,a_4=2\,a_0a_3+2\,a_1a_2$$

Deoarece $y(0)=1$ rezultă $a_0=1$ şi $a_1=1$, $a_2=1$

$$a_3=\frac{4}{3},\ a_4=\frac{7}{6}$$

După înlocuirea acestor coeficienţi în prima expresie din (10) se obţine

$$y(x)=1+x+x^2+\frac{4}{3}x^3+\frac{7}{6}x^4 \tag{11}$$

din (11) se vede că s-a obţinut aceeaşi expresie pentru soluţie ca şi în metoda Taylor dată prin (8), dar de această dată nu a mai fost necesară evaluarea derivatelor pînă la ordinul $p=4$, din dezvoltarea în serie Taylor. Trebuie menţionat că metoda Taylor şi metoda coeficienţilor nedeterminaţi pot fi aplicate şi la ecuaţii diferenţiale de ordin superior, dar cu o serie de condiţii în plus.

16.3. METODA LUI EULER (METODA LINIILOR POLIGONALE)

Este una din metodele cele mai simple pentru rezolvarea ecuaţiilor diferenţiale ordinare. Algoritmul metodei rezultă, fie utilizînd dezvoltatea în serie Taylor (4), fie dintr-o interpretare geometrică (datorită acestei interpretări geometrice i se mai spune şi metoda liniilor poligonale).

Dacă din dezvoltarea în serie Taylor (4) se reţin numai primii doi termeni se obţine

$$y_1=y_0+hf(x_0, y_0) \tag{12}$$

Interpretarea geometrică a relaţiei (12) constă în a înlocui curba $y=y(x)$ cu tangenta ei în punctul (x_0, y_0) pe intervalul, $h=x_1-x_0$. Îndată ce y_1 a fost calculat, el poate fi utilizat în locul lui y_0 în (12) pentru calculul lui y_2, adică

$$y_2=y_1+hf(x_1, y_1) \tag{13}$$

În general pentru orice şir $x_0<x_1<x_2<\ldots$ se poate determina prin metoda lui Euler un şir de valori aproximative $y_1, y_2, y_3, \ldots$ valori ce aproximează soluţia numerică exactă $y(x_1), y(x_2), y(x_3), \ldots$, plecînd de la valoarea iniţială y_0.

Şirul de valori aproximative menţionat se pot calcula cu ajutorul relaţiei iterative

$$y_{i+1}=y_i+hf(x_i, y_i) \quad i=0, 1, 2, \ldots \tag{14}$$

Dacă se unesc prin segmente de dreaptă punctele (x_0, y_0), (x_1, y_1) (x_2, y_2), ... se obţine o linie poligonală întîlnită în literatură sub denumirea de „poligonul lui Euler", (vezi fig. 16.1). Poligoanele lui Euler pot fi utilizate la demonstrarea existenţei soluţiei ecuaţiei diferenţiale, dacă $f(x, y)$ este continuă în domeniul considerat, de asemenea poligonul lui Euler poate fi folosit la demonstrarea existenţei şi unicităţii soluţiei ecuaţiei diferenţiale dacă $f(x, y)$ este continuă şi satisface condiţia Lipschitz în variabila dependentă y. În continuare se va prezenta programul în BASIC care foloseşte algoritmul metodei Euler dat în (14) pentru găsirea soluţiei numerice a ecuaţiei diferenţiale $y'(x)=x+y^2$, cu $y(0)=1$, $h=0,1$ pe intervalul $0\leqslant x\leqslant 1$.

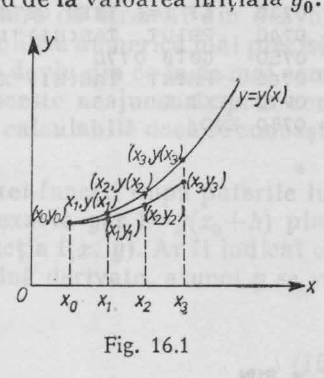


Fig. 16.1

```
0001 REM PRØGRAM P2 CAP 16.
0010 PRINT "PRØGRAMUL BASIC PT.ALGØRITMUL METØDEI EULER DAT IN(14)"

0015 PRINT "PT. GASIREA SØLUTIEI NUMERICE A EC. DIFERENTIALE"
0016 PRINT "       Y'=X+Y↑2 CU CØNDITIA INIYIALA Y(0)=1"
0017 PRINT "              SI  PASUL H=0.1"
0018 PRINT "       PE INTERVALUL 0<=X<=1"
0038 INPUT P2,B6,H1
0040 READ X0,Y0,H
0041 LET N=B6/H+1
0042 DIM V[N,1]
0043 LET I=1
0045 PRINT "X0="X0,"Y0="Y0,"H="H
0050 LET F=X0+Y0↑2
0060 LET Y1=Y0+H*F
0062 IF I>N THEN GØTØ 0070
0063 LET V[I,1]=Y0
0064 LET I=I+1
0070 PRINT "X="X0,"H="H,"F="F,"Y="Y1
0080 LET X0=X0+H
0090 IF X0>1 THEN GØTØ 0590
0100 LET Y0=Y1
0110 GØTØ 0050
0120 DATA 0,1,.1
0130 END
0590 FØR I=1 TØ N
0600   IF I<>1 THEN GØTØ 0680
0610   FØR J=H1 TØ 70
0612     IF J=H1 THEN PRINT  TAB(J)"0";
0615     IF J=70 THEN PRINT  TAB(J)"Y"
0620     LET A=V[I,1]/P2+H1
0630     IF J=A THEN GØTØ 0660
0640     PRINT TAB(J)"-";
0650     GØTØ 0670
0660     PRINT  TAB(J)"*";
0670   NEXT J
0675   PRINT
0680   LET S=V[I,1]/P2
0690   LET S=S+H1
0700   PRINT  TAB(H1)"!"; TAB(S)"*"
0710 NEXT I
0720 FØR I=1 TØ 6
0730   IF I=6 THEN GØTØ 0760
0740   PRINT  TAB(H1)"!"
0750   GØTØ 0770
0760   PRINT  TAB(H1)"X"
0770 NEXT I
0780 END
*

* RUN
PRØGRAMUL BASIC PT.ALGØRITMUL METØDEI EULER DAT IN(14)
PT. GASIREA SØLUTIEI NUMERICE A EC. DIFERENTIALE
   Y'=X+Y↑2 CU CØNDITIA INIYIALA Y(0)=1
          SI  PASUL H=0.1
   PE INTERVALUL 0<=X<=1
```

```
 ? .2 ? 1 ? 5
X0= 0        Y0= 1       H= .1
X= 0         H= .1       F= 1          Y= 1.1
X= .1        H= .1       F= 1.31       Y= 1.231
X= .2        H= .1       F= 1.71536    Y= 1.40254
X= .3        H= .1       F= 2.26711    Y= 1.62925
X= .4        H= .1       F= 3.05445    Y= 1.93469
X= .5        H= .1       F= 4.24304    Y= 2.359
X= .6        H= .1       F= 6.16487    Y= 2.97549
X= .7        H= .1       F= 9.55352    Y= 3.93084
X= .8        H= .1       F= 16.2515    Y= 5.55599
X= .9        H= .1       F= 31.769     Y= 8.73289
X= 1         H= .1       F= 77.2633    Y= 16.4592
    0-----*-------------------------------------------------------------Y
    !    *
    !    *
    !     *
    !       *
    !        *
    !         *
    !           *
    !              *
    !                   *
    !                        *
    !
    !
    !
    !
    !
    X

END AT 0780
*
```

16.4. METODA GENERALĂ RUNGE-KUTTA

Fie ecuaţia diferenţială

$$y'=f(x,\ y) \text{ cu condiţia iniţială } y(x_0)=y_0$$

S-a arătat anterior că reţinerea unui număr mare de termeni din dezvoltarea în serie Taylor (4), (lucru ce ar conduce la o soluţie numerică mai precisă) este aproape imposibilă, din cauza derivărilor care devin din ce în ce mai complicate. Metoda Runge-Kutta caută să înlăture aceste neajunsuri, prin construcţia unei funcţii de h, de o anumită formă, uşor calculabilă dacă se cunoaşte $f(x, y)$.

Se pune condiţia ca dezvoltarea în serie a acestei funcţii după puterile lui n să coincidă cu dezvoltarea în serie a soluţiei exacte $y(x_1)=y(x_0+h)$ pînă la o putere cît mai mare a lui h independent de funcţia $f(x, y)$. Ar fi indicat ca expresia lui y_1 (aproximaţia lui $y(x_1)$) să nu conţină derivate, atunci y se va exprima sub forma

$$y_1=y_0+\sum_{j=1}^{s} P_{sj}k_j \tag{15}$$

unde s întreg, P_{sj} sînt constante care trebuiesc determinate, k_j sînt valori ale lui $f(x, y)$ în puncte vecine punctului (x_0, y_0) valori amplificate cu pasul de discretizare h, adică

$$K_j=hf(\zeta_j, \eta_j) \qquad \zeta_j=x_0+\alpha_j h$$
$$(j=1, 2, ..., s) \tag{16}$$
$$j=1, 2, ..., s \qquad \eta_j=y_0+\sum_{i=1}^{j-1} \beta_{ji} k_i$$

unde α_j, β_j, P_{sj} fiind constante.

În toate cazurile se consideră că

$$\alpha_1=0, \beta_{01}=0, \zeta_1=x_0, \eta_1=y_0$$

Pentru $s=1, 2, 3$, se obţine

$$\left.\begin{array}{l} k_1=hf(x_0, y_0); \; k_2=hf(x_0+\alpha_2 h, y+\beta_{21}k_1); \\ k_3=hf(x+\alpha_3 h, y_0+\beta_{31}k_1+\beta_{32}k_2); \end{array}\right\} \tag{17}$$

Din (17) se vede clar caracterul iterativ al procedeului Runge-Kutta. Rămîn de determinat constantele P_{sj}, α_j şi β_{ji}, din condiţia ca cele două dezvoltări în serie după puterile lui h, date în (4) şi (15) să coincidă pînă la o anumită putere a lui h, independent de funcţia f(x, y). Pentru diverse valori date lui s în (15) se obţin diverse formule de tip Runge-Kutta. Pentru $s=1$ se găseşte

$$y_1=y_0+hf(x_0, y_0) \tag{17'}$$

adică metoda lui Euller

Pentru $s=2$, din formulele (15) şi (16) rezultă

— *Metoda lui* EULER *îmbunătăţită*

$$y_1=y_0+\sum_{j=1}^{2} P_{2j},$$

sau sub forma dezvoltată

$$y=y_0+P_{21}hf(x_0, y_0)+P_{22}hf(x_0+\alpha_2 h, y_0+\beta_{21}hf(x_0, y_0)) \tag{18}$$

De asemenea dezvoltarea în serie a soluţiei analitice $y(x)$ în punctul $x=x_1$ şi reţinînd numai primii 4 termeni rezultă.

$$y(x_1)=y_0+hy'(x_0)+\frac{h^2}{2!}y''(x_0)+\frac{h^3}{3!}y'''(x_0+\theta x) \quad {\scriptstyle 0<\theta<1} \tag{19}$$

Dacă se exprimă cu ajutorul lui f(x, y) calculată în (x_0, y_0) derivatele lui y din (19), atunci (19) devine

$$y(x_1)=y_0+hf(x_0, y_0)+\frac{h^2}{2!}[f'_x(x_0, y_0)+f'_y(x_0, y_0)\cdot f(x_0, y_0)]+T_2 \tag{20}$$

unde $T_2=\frac{h^3}{3!}y'''(x+\theta x)$

Dezvoltarea lui y_1 dată în (18) după puterile lui h va fi

$$y_1=y_0+P_{21}hf(x_0, y_0)+P_{22}h[f(x_0, y_0)+\alpha_2 hf'_x(x_0y_0)+\beta_{21}hf(x_0y_0)\cdot f_y(x_0, y_0)] \tag{21}$$

Punîndu-se condiţia ca expresiile (20) şi (21) să coincidă pînă la termenul al treilea cel care conţine pe h^2, independent de $f(x, y)$, se ajunge la următorul sistem pentru determinarea constantelor.

$$P_{21}+P_{22}=1,\ P_{22}\alpha_2=1/2,\ P_{22}\beta_{21}=1/2 \tag{22}$$

Din sistemul (22) se vede că $P_{22}\neq 0$ este o necesitate şi că soluţia sistemului nu este unică.

Pentru alegerea $P_{22}=1/2$ se obţine

$P_{21}=1/2$, $\alpha_2=1$, $\beta_{21}=1$, cu aceşti coeficienţi y_1 din (21) devine

$$y_1=y_0+\frac{1}{2}(k_1+k_2) \tag{23}$$

unde după înlocuirea lui k_1 şi k_2 (23) devine

$$y_1=y_0+\frac{h}{2}[\mathrm{f}(x_0,\ y_0)+\mathrm{f}(x_0+h,\ y_0+h\mathrm{f}(x_0,\ y_0))] \tag{24}$$

După determinarea lui y_1 cu ajutorul relaţiei (24) se poate calcula y_2 cu ajutorul relaţiei

$$y_2=y_1+\frac{h}{2}[\mathrm{f}(x_1,\ y_1)+\mathrm{f}(x_1+h,\ y_1+hf(x_1,\ y_1))]$$

Dacă se continuă în acest fel se poate calcula şirul de aproximaţii y_1, y_2, ..., y_n cu ajutorul procesului iterativ următor

$$y_{i+1}=y_i+\frac{h}{2}[\mathrm{f}(x_i,\ y_i)+\mathrm{f}(x_i+h,\ y_i+h\mathrm{f}(x_i,\ y_i))] \tag{25}$$

$$i=0,\ 1,\ 2,\ ...,\ n$$

Algoritmul definit prin (25) constituie algoritmul (metodei Euler îmbunătăţite) pentru generarea aproximaţiilor y_1, y_2, ..., y_n ale valorilor exacte $y(x_1)$, $y(x_2)$, ... ,$y(x_n)$, plecînd de la valoarea cunoscută y_0.

Acest algoritm a rezultat din procedeul Runge-Kutta pentru $s=2$.

Pentru $s=3$ se obţine într-o manieră asemănătoare metoda Runge-Kutta de ordinul 3

16.5. METODA RUNGE-KUTTA DE ORDINUL TREI

În acest caz se obţine

$$k_1=hf(x_0,\ y(x_0)) \tag{26}$$

$$k_2=hf\left(x_0+\frac{h}{2},\ y(x_0)+\frac{1}{2}k_1\right)$$

$$k_3=hf(x_0+h,\ y(x_0)+2k_2-k_1)$$

iar algoritmul de calcul este dat de relaţia

$$y_{i+1}=y_i+\frac{1}{6}(k_1+4\,k_2+k_3) \quad i=0, 1, 2, \ldots, \tag{27}$$

Din (27) se vede că metoda Runge-Kutta de ordinul trei este analoagă cu formula de integrare Simpson.

Cazul cel mai des utilizat în practică este pentru s=4, care conduce la determinarea metodei Runge-Kutta de ordinul patru.

16.6. METODEI RUNGE-KUTTA DE ORDINUL PATRU

Se foloseşte raţionamentul de la $s=2$, pentru determinarea constantelor P_{sj}, α_j, β_{ji} ajungîndu-se la un sistem de 10 ecuaţii cu 13 necunoscute, care nu are soluţie unică.

Pentru sistemul obţinut s-a determinat diferite sisteme de constante care să verifice sistemul alegîndu-se un set de valori cu structură simplă pentru uşurinţa calculelor. Soluţia sistemului care satisface dezideratul de mai sus conduce la următorul algoritm: [47]:

$$k_1=hf(x_i, y_i);\quad k_2=hf\left(x_i+\frac{h}{2},\ y_i+\frac{k_1}{2}\right) \tag{28}$$

$$k_3=hf\left(x_i+\frac{h}{2}, y_i+\frac{k_2}{2}\right);\quad k_4=hf(x_i+h,\ y_i+k_3)$$

iar

$$Y_{i+1}=y_i+\frac{1}{6}(k_1+2\,k_2+2\,k_3+k_4),\ i=0, 1, \ldots, n \tag{29}$$

Principalul avantaj al metodei Runge-Kutta, şi în general al tuturor metodelor cu paşi separaţi, îl constituie faptul că pentru calculul unei noi aproximări e suficientă cunoaşterea unui singur punct anterior.

Acest avantaj împreună cu o precizie destul de bună pe care o oferă metoda Runge-Kutta cu $s=4$, a făcut ca metoda să fie foarte mult utilizată în practică pentru determinarea valorilor iniţiale necesare metodelor cu paşi legaţi de tip predictor-corector. Metoda prezintă şi un dezavantaj, deoarece pentru fiecare aproximaţie, necesită evaluarea funcţiei $f(x, y)$ pentru mai multe valori ale lui x, ceea ce implică creşterea timpului de calcul.

În continuare se vor prezenta două programe scrise în BASIC pentru găsirea soluţiei numerice a ecuaţiei $y'=-y$ cu condiţia iniţială $y(0)=1$, cu $h=0{,}5$ pe intervalul $0\leqslant x\leqslant 5$.

Primul program în BASIC codifică algoritmul dat prin relaţiile (26) şi (27) pentru metoda Runge-Kutta de ordinul trei, iar al doilea program codifică algoritmul de calcul dat prin relaţiile (28) şi (29) pentru metoda Runge-Kutta de ordinul patru.

Ambele programe s-au aplicat la aceeaşi ecuaţie diferenţială ordinară pentru a se putea analiza precizia celor două algoritme de calcul ale metodei Runge-Kutta.

```
0005 REM PRØGRAM P3 CAP.16.
0010 PRINT "ALGØRITMUL PENTRU METØDA RUNGE-KUTTA DE ØRDINUL 3"
0012 PRINT "DAT PRI N RELATIILE (26) SI (27)"
0014 PRINT "PENTRU GASIREA SØLUTIEI NUMERICE A EC. Y'=-Y"
0015 PRINT "              CU CØNDITIA INITIALA Y0=1"
0016 PRINT "CU PASUL H=0,5 PE INTERVALUL 0<=X<=5"
0030 READ X0,Y0,H
0040 PRINT "X0="X0,"Y0="Y0,"H="H
0050 LET K1=H*(-Y0)
0060 LET K2=H*(-Y0+K1/2)
0070 LET K3=H*(-Y0+2*K2-K1)
0080 LET Y=Y0+1/6*(K1+4*K2+K3)
0090 PRINT "K1="K1,"K2="K2,"K3="K3,"Y="Y
0100 LET X0=X0+H
0105 LET Y0=Y
0110 IF X0<=5 THEN GØTØ 0050
0120 DATA 0,1,.5
0130 END

* RUN
ALGØRITMUL PENTRU METØDA RUNGE-KUTTA DE ØRDINUL 3
DAT PRI N RELATIILE (26) SI (27)
PENTRU GASIREA SØLUTIEI NUMERICE A EC. Y'=-Y
          CU CØNDITIA INITIALA Y0=1
CU PASUL H=0,5 PE INTERVALUL 0<=X<=5
X0= 0           Y0= 1           H= .5
K1=-.5          K2=-.625        K3=-.875        Y= .354167
K1=-.177083     K2=-.221354     K3=-.309896     Y= .125434
K1=-.062717     K2=-7.83963E-02                 K3=-.109755     Y= 4.44245E-02
K1=-2.22123E-02                 K2=-2.77653E-02                 K3=-3.88715E-02
Y= 1.57337E-02
K1=-7.86684E-03                 K2=-9.83356E-03                 K3=-.013767
Y= 5.57235E-03
K1=-2.78617E-03                 K2=-3.48272E-03                 K3=-4.87581E-03
Y= 1.97354E-03
K1=-9.86768E-04                 K2=-1.23346E-03                 K3=-1.72685E-03
Y= 6.9896E-04
K1=-3.4948E-04                  K2=-4.3685E-04                  K3=-6.1159E-04
Y= 2.47548E-04
K1=-1.23774E-04                 K2=-1.54718E-04                 K3=-2.16605E-04
Y= 8.76733E-05
K1=-4.38366E-05                 K2=-5.47958E-05                 K3=-7.67142E-05
Y= 3.1051E-05
K1=-1.55255E-05                 K2=-1.94069E-05                 K3=-2.71696E-05
Y= 1.09972E-05

END AT 0130
*
```

```
0005 REM PRØGRAM P4 CAP.16.
0010 PRINT "ALGØRITMUL PTR. METØDA RUNGE-KUTTA DE ØRDINUL 4"
0015 PRINT "DETERMINAT PRIN RELATIILE (28) SI (29)"
0016 PRINT "PENTRU GASIREA SØLUTIEI NUMERICE A EC.Y'=-Y"
0017 PRINT "CU CØNDITIA INITIALA Y(0)=1"
0018 PRINT "CU PASUL =0.5 PE INTERVALUL 0<=X<=5"
0030 READ X0,Y0,H
0040 PRINT "X0="X0,"Y0="Y0,"H="H
0050 LET K1=H*(-Y0)
0060 LET K2=H*(-Y0+K1/2)
0070 LET K3=H*(-Y0+K2/2)
```

```
0080 LET K4=H*(-Y0+K3)
0090 LET Y=Y0+1/6*(K1+2*K2+2*K3+K4)
0100 PRINT "K1="K1,"K2="K2,"K3="K3,"K4="K4,"Y="Y
0105 LET X0=X0+H
0107 LET Y0=Y
0110 IF X0<=5 THEN GØTØ 0050
0120 DATA 0,1,.5
0130 END

* RUN
ALGØRITMUL PTR. METØDA RUNGE-KUTTA DE ØRDINUL 4
DETERMINAT PRIN RELATIILE (28) SI (29)
PENTRU GASIREA SØLUTIEI NUMERICE A EC.Y'=-Y
CU CØNDITIA INITIALA Y(0)=1
CU PASUL =0.5 PE INTERVALUL 0<=X<=5
X0= 0            Y0= 1            H= .5
K1=-.5           K2=-.625         K3=-.65625       K4=-.828125   Y= .351562
K1=-.175781      K2=-.219727      K3=-.230713      K4=-.291138   Y= .123596
K1=-6.17981E-02                   K2=-7.72476E-02                K3=-.08111
K4=-.102353      Y= 4.34517E-02
K1=-2.17259E-02                   K2=-2.71573E-02                K3=-2.85152E-02
K4=-3.59835E-02                   Y= .015276
K1=-.007638      K2=-9.5475E-03                    K3=-1.00249E-02
K4=-1.26504E-02                   Y= 5.37046E-03
K1=-2.68523E-03                   K2=-3.35654E-03                K3=-3.52437E-03

K4=-4.44742E-03                   Y= 1.88805E-03
K1=-9.44026E-04                   K2=-1.18003E-03                K3=-1.23903E-03
K4=-1.56354E-03                   Y= 6.63769E-04
K1=-3.31884E-04                   K2=-4.14855E-04                K3=-4.35598E-04
K4=-5.49683E-04                   Y= 2.33356E-04
K1=-1.16678E-04                   K2=-1.45847E-04                K3=-1.5314E-04
K4=-1.93248E-04                   Y= 8.20391E-05
K1=-4.10195E-05                   K2=-5.12744E-05                K3=-5.38382E-05
K4=-6.79386E-05                   Y= 2.88419E-05
K1=-1.44209E-05                   K2=-1.80262E-05                K3=-1.89275E-05
K4=-2.38847E-05                   Y= 1.01397E-05

END AT 0130
*
```

16.7. METODA LUI EULER CU PREDICŢIE ŞI CORECŢIE

Relaţiile (14) şi (25), care constituie algoritmul metodei Euler respectiv Euler îmbunătăţită, pentru generarea şirului de aproximaţii $\{y_i\}_{i\in N}$ asociate şirului de valori exacte $\{y(x_i)\}_{i\in N}$ pot fi utilizate împreună şi conduc la un algoritm de tip implicit.

Cu ajutorul relaţiei (14) se obţine o valoare prescrisă sau cu predicţie pentru y_{i+1}, datorită acestui fapt (14) se scrie sub forma

$$y_{i+1}^{p}=y_i+hf(x_i, y_i),\ i=0, 1, 2, \dots, n \tag{30}$$

Valoarea lui y_{i+1}^{P} obţinută din (30) se introduce pentru recalcularea unei valori mai precise pentru y_{i+1}, notată prin y_{i+1}^{C}, scrisă sub forma

$$y_{i+1}^{C}=y_i+\frac{h}{2}[f(x_i, y_i)+f(x_{i+1}, y_{i+1}^{P})]\quad i=0, 1, 2, \dots, n \tag{31}$$

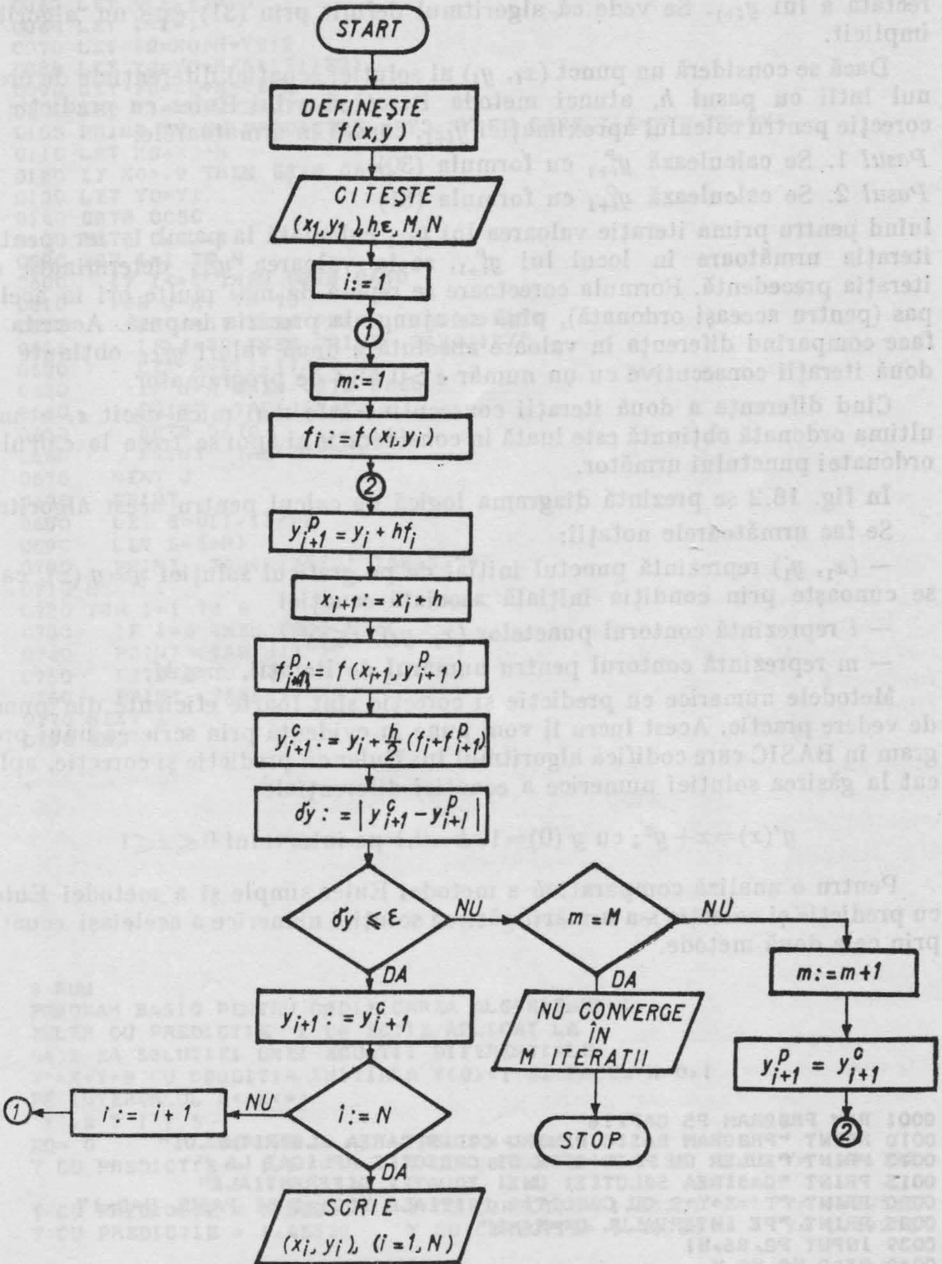

START
DEFINEŞTE f(x,y)
CITEŞTE (x1,y1), h, ε, M, N
i:=1
1
m:=1
fi := f(xi, yi)
2
yp i+1 = yi + h fi
xi+1 := xi + h
fp i+1 := f(xi+1, yp i+1)
yc i+1 := yi + h/2 (fi + fp i+1)
δy := | yc i+1 − yp i+1 |
δy ≤ ε
NU
m = M
NU
m:=m+1
yp i+1 = yc i+1
2
DA
NU CONVERGE ÎN M ITERAŢII
STOP
DA
yi+1 := yc i+1
i := N
NU
i := i+1
1
DA
SCRIE (xi, yi), (i = 1, N)
STOP

Fig. 16.2

Relaţia (31) se numeşte formula corectoare, iar y_{i+1}^{C} reprezintă o valoare corectată a lui y_{i+1}. Se vede că algoritmul definit prin (31) este un algoritm implicit.

Dacă se consideră un punct (x_i, y_i) al soluţiei ecuaţiei diferenţiale de ordinul întîi cu pasul h, atunci metoda iterativă a lui Euler cu predicţie şi corecţie pentru calculul aproximaţiei y_{i+1}, constă în următoarele:

Pasul 1. Se calculează y_{i+1}^{P} cu formula (30)

Pasul 2. Se calculează y_{i+1}^{C} cu formula (31)

luînd pentru prima iteraţie valoarea lui y_{i+1} calculată la pasul 1, iar pentru iteraţia următoare în locul lui y_{i+1}^{P}, se ia valoarea y_{i+1}^{C} determinată de iteraţia precedentă. Formula corectoare se repetă de mai multe ori la acelaş pas (pentru aceeaşi ordonată), pînă se ajunge la precizia impusă. Aceasta se face comparînd diferenţa în valoare absolută a două valori y_{i+1}^{C} obţinute în două iteraţii consecutive cu un număr $\varepsilon>0$ ales de programator.

Cînd diferenţa a două iteraţii consecutive este mai mică decît ε, atunci ultima ordonată obţinută este luată în consideraţie şi apoi se trece la calculul ordonatei punctului următor.

În fig. 16.2 se prezintă diagrama logică de calcul pentru acest algoritm.

Se fac următoarele notaţii:

— (x_1, y_1) reprezintă punctul iniţial de pe graficul soluţiei $y=y\ (x)$, care se cunoaşte prin condiţia iniţială asociată ecuaţiei

— i reprezintă contorul punctelor (x_i, y_i), $i\leqslant N$

— m reprezintă contorul pentru numărul de iteraţii, $m\leqslant M$.

Metodele numerice cu predicţie şi corecţie sînt foarte eficiente din punct de vedere practic. Acest lucru îl vom pune în evidenţă prin scrierea unui program în BASIC care codifică algoritmul lui Euler cu predicţie şi corecţie, aplicat la găsirea soluţiei numerice a ecuaţiei diferenţiale:

$$y'(x)=x+y^2;\ \text{cu}\ y\ (0)=1,\ h=0{,}1\ \text{pe intervalul}\ 0\leqslant x\leqslant 1$$

Pentru o analiză comparativă a metodei Euler simple şi a metodei Euler cu predicţie şi corecţie s-a urmărit găsirea soluţiei numerice a aceleiaşi ecuaţii prin cele două metode.

```
0001 REM PRØGRAM P5 CAP.16
0010 PRINT "PRØGRAM BASIC PENTRU CØDIFICAREA ALGØRITMULUI"
0013 PRINT "EULER CU PREDICTIE SI CØRECTIE APLICAT LA "
0015 PRINT "GASIREA SØLUTIEI UNEI ECUATII DIFERENTIALE"
0020 PRINT "Y'=X+Y↑2 CU CØNDITIA INITIALA Y(0)=1 SI PASUL H=0.1"
0025 PRINT "PE INTERVALUL 0<=X<=1"
0039 INPUT P2,B6,H1
0040 READ X0,Y0,H
0041 LET N=B6/H+1
0042 DIM V[N,1]
0043 LET I=1
0045 PRINT "X0="X0,"Y0="Y0,"PASUL H="H
0050 LET F1=X0+Y0↑2
0060 LET Y3=Y0+H*F1
```

```
0062 IF I>N THEN GØTØ 0070
0063 LET V[I,1]=Y0
0064 LET I=I+1
0070 LET F2=X0+H+Y3↑2
0080 LET Y2=Y0+H/2*(F1+F2)
0090 LET F0=X0+H+Y2↑2
0100 LET Y1=Y0+H/2*(F1+F0)
0105 PRINT "Y CU PREDICTIE ="Y3,"Y CU CØRECTIE="Y2,"Y="Y1
0110 LET X0=X0+H
0120 IF X0>.9 THEN GØTØ 0590
0130 LET Y0=Y1
0140 GØTØ 0050
0150 DATA 0,1,.1
0590 FØR I=1 TØ N
0600   IF I<>1 THEN GØTØ 0680
0610   FØR J=H1 TØ 70
0612     IF J=H1 THEN PRINT  TAB(J)"0";
0615     IF J=70 THEN PRINT  TAB(J)"Y"
0620     LET A=V[I,1]/P2+H1
0630     IF J=A THEN GØTØ 0660
0640     PRINT  TAB(J)"-";
0650     GØTØ 0670
0660     PRINT  TAB(J)"*";
0670   NEXT J
0675   PRINT
0680   LET S=V[I,1]/P2
0690   LET S=S+H1
0700   PRINT  TAB(H1)"I"; TAB(S)"*"
0710 NEXT I
0720 FØR I=1 TØ 6
0730   IF I=6 THEN GØTØ 0760
0740   PRINT  TAB(H1)"!"
0750   GØTØ 0770
0760   PRINT  TAB(H1)"X"
0770 NEXT I
0780 END

*



* RUN
PRØGRAM BASIC PENTRU CØDIFICAREA ALGØRITMULUI
EULER CU PREDICTIE I CØ ECTIE APLICAT LA
GA I EA SØLUTIEI UNEI ECUATII DIFERENTIALE
Y'=X+Y↑2 CU CØNDITIA INITIALA Y(0)=1 SI PASUL H=0,1
PE INTERVALUL 0<=X<=1
 ? .2 ? 1 ? 5
X0= 0           Y0= 1           PASUL H= .1
Y CU PREDICTIE = 1.1         Y CU CØRECTIE= 1.1155         Y= 1.11722

Y CU PREDICTIE = 1.25203     Y CU CØRECTIE= 1.27301        Y= 1.27565
Y CU PREDICTIE = 1.45838     Y CU CØRECTIE= 1.48836        Y= 1.49278

Y CU PREDICTIE = 1.74562     Y CU CØRECTIE= 1.79156        Y= 1.79968
Y CU PREDICTIE = 2.16357     Y CU CØRECTIE= 2.24068        Y= 2.25766
Y CU PREDICTIE = 2.81736     Y CU CØRECTIE= 2.96438        Y= 3.00689
Y CU PREDICTIE = 3.97102     Y CU CØRECTIE= 4.31241        Y= 4.4538
Y CU PREDICTIE = 6.50743     Y CU CØRECTIE= 7.63794        Y= 8.43752
Y CU PREDICTIE = 15.6367     Y CU CØRECTIE= 24.3074        Y= 41.6247
```

```
O---*--------------------------------------------------------------------------------------Y
!   *                                                                                      -
!   *
!    *
!     *
!       *
!          *
!              *
!                   *
!                             *
!*
!
!
!
!
!
X

END AT 0780
*
```

16.8. METODA PREDICTOR-CORECTOR CU PAŞI LEGAŢI A LUI ADAMS.

Se consideră ecuaţia diferenţială de ordinul întîi:

$$y'=\mathrm{f}\,(x,\ y) \text{ cu condiţia iniţială } y\,(x_0)=y \tag{32}$$

Metoda Runge-Kutta este una din cele mai eficiente metode cu paşi separaţi, dar are dezavantajul că pentru calculul fiecărei ordonate este necesară evaluarea funcţiei f (x, y) pentru mai multe valori ale lui x, fapt ce implică creşterea timpului de calcul.

Metodele predictor-corectoare cu paşi legaţi sînt din acest punct de vedere mult mai avantajoase, dar cu dezavantajul că nu pot să-şi determine valorile iniţiale de start, pentru înlăturarea acestui inconvenient la început se foloseşte o metodă cu paşi separaţi care determină toate valorile de start necesare lansării algoritmului predictor-corector cu paşi legaţi.

Presupunînd că se foloseşte metoda Runge-Kutta de ordinul patru la determinarea soluţiei numerice în punctele $x_0, x_1, x_2, \ldots, x_n$ cu $x_i=x+ih$

$$i=1,\ 2,\ \ldots,\ n \tag{33}$$

adică

$$y_0,\ y_1,\ y_2,\ \ldots,\ y_n$$

şi se cere determinarea în continuare prin altă metodă a aproximaţiei y_{n+1}.

Pentru aceasta se va utiliza metoda predictor-corectoare a lui Adams, care are la bază relaţia

$$y\,(x_{n+1})-y(x_n)=\int_{x_n}^{x_{n+1}} \mathrm{f}\,(x,\ y\,(x))\,\mathrm{d}x \tag{34}$$

Datorită faptului că $y(x)$ este soluţia ecuaţiei diferenţiale (32) şi fiind necunoscută nu se cunoaşte nici expresia analitică a funcţiei $\mathrm{F}(x)=\mathrm{f}(x,\ y\,(x))$ din partea dreaptă a lui (34) (deoarece F (x) este funcţie de y) totuşi se cunosc valorile sale pentru x egal cu

$$x_n,\ x_{n-1},\ \ldots,\ x_{n-k},\ k\leqslant n$$

Formula predictoare se obţine înlocuind în (34) funcţia F (x)=f $(x,y(x))$ printr-un polinom de interpolare Newton de speţa doua, definit pentru nodurile x_n, x_{n-1}, x_{n-k}. Cu notaţia $f_n=f(x_n,\ y(x_n))$ se obţine

$$y_{n+1}^{P}=y_n+\int_{x_n}^{x_{n+1}}\left[f_n+\frac{\nabla f_n}{1!h}(x-x_n)+\frac{\nabla^2 f_n}{2!h^2}(x-x_n)(x-x_{n-1})+\ldots\right.$$

$$\left.\ldots+\frac{\nabla^k f_n}{k!h^k}(x-x_n)(x-x_{n-1})\ldots(x-x_{n-k+11})\right]dx$$

Dacă se face schimbarea de variabilă $u=\frac{x-x_n}{h}$ se obţine:

$$y_{n+1}^{P}=y_n+n\left[f_n+\frac{\nabla f_n}{2}+\frac{5}{12}\nabla^2 f_n+\frac{3}{8}\Delta^3 f_n+\frac{251}{720}\Delta^4 f_n+\ldots\right], \qquad (36)$$

relaţie ce reprezintă formula predictoare a metodei.

Cu ajutorul formulei (36) se poate calcula

$$f_{n+1}=f(x_{n+1},\ y_{n+1}^{P})$$

În aceeaşi manieră se poate construi un nou polinom Newton de speţa a doua şi gradul k, utilizînd nodurile x_{n+1}, x_n, ..., x_{n+1-k} şi valorile funcţiei f $(x,\ y)$ în aceste noduri. Înlocuind funcţia f $(x,\ y)$ din (34) prin polinomul de interpolare astfel construit, se obţine

$$y_{n+1}^{C}=y_n+\int_{x_n}^{x_{n+1}}\left[f_{n+1}+\frac{\nabla f_{n+1}}{1!h}(x-x_{n+1})+\frac{\nabla^2 f_{n+1}}{2!h^2}(x-x_{n+1})(x-x_n)+\ldots\right.$$

$$\left.\ldots+\frac{\nabla^k f_{n+1}}{k!h^k}(x-x_{n+1})(x-x_n)\ldots(x-x_{n-k+2})\right]dx$$

Dacă se face schimbarea de variabile $u=\frac{x-x_{n+1}}{h}$ şi apoi se integrează rezultă formula corectoare a metodei:

$$y_{n+1}^{C}=y_n+h\left[f_{n+1}-\frac{1}{2}\Delta f_{n+1}-\frac{1}{12}\Delta^2 f_{n-1}-\frac{1}{24}\Delta^3 f_{n+1}-\frac{1}{720}\Delta f_{n+1}+\ldots\right]$$

Considerînd diverse valori pentru k în formula predictor, respectiv corector şi ţinînd seama de relaţia:

$$\Delta^m f_{n+i}=f_{n+i}-C_m^1 f_{n+i-1}+C_m^2 f_{n+i-2}+\ldots+(-1)^m C_m^m f_{n+i-m};$$

$$m=1,\ 2,\ \ldots,\ k;\quad i=0,1$$

se obţin diverse cazuri particulare ale formulelor predictor corectoare. Astfel avem

$$k=1\left\{\begin{aligned} y_{n+1}^{P}&=y_n+\frac{h}{2}[3f_n-f_{n-1}]\\ y_{n+1}^{C}&=y_n+\frac{h}{2}[f_{n+1}^{P}+f_n]\end{aligned}\right\}\ n=1,\ 2,\ \ldots \qquad (38)$$

$$k=2 \quad \begin{aligned} y^{P}_{n+1} &= y_n + \frac{h}{12}[23\,f_n - 16\,f_{n-1} + 5\,f_{n-2}] \\ y^{C}_{n+1} &= y_n + \frac{h}{12}[5\,f_{n+1} + 8\,f_n - f_{n-1}] \end{aligned} \quad n=2, 3, \ldots \tag{39}$$

$$k=3 \quad \begin{aligned} y^{P}_{n+1} &= y_n + \frac{h}{24}[55\,f_n - 59\,f_{n-1} + 37\,f_{n-2} - 9\,f_{n-3}] \\ y^{C}_{n+1} &= y_n + \frac{h}{24}[9\,f_{n+1} + 19\,f_n - 5\,f_{n-1} + f_{n-2}] \end{aligned} \quad n=3, \ldots \tag{40}$$

Din algoritmul prezentat se vede că această metodă este iterativă și valorile de start se vor calcula cu ajutorul metodei Runge-Kutta, sau al altei metode cu pași separați. [47]

În continuare se vor prezenta 3 programe în BASIC care vor folosi în prima parte metoda Runge-Kutta pentru determinarea mărimilor de start din algoritmele date în (38), (39) și (40):

— Primul program va folosi algoritmul dat în (38)

— Al doilea program va folosi algoritmul dat în (39)

— Iar al treilea va folosi algoritmul dat în (40).

Toate trei programele se vor utiliza la determinarea soluției numerice a ecuației diferențiale ordinare $y'=x+y^2$, cu $y(0)=1$, pentru $h=0,2$ pe intervalul $0 \leqslant x \leqslant 1$ după care se poate face o analiză asupra rezultatelor obținute prin cele trei algoritme.

```
0010 REM PRØGRAM P6. CAP.16
0020 PRINT "METØDA PREDICTØR-CØRECTØR CU PASI LEGATI A LUI ADAMS"
0030 PRINT "FØLØSIND ALGØRITMUL DAT IN (38)"
0040 PRINT "ECUATIA DIFERENTIALA ESTE Y'=X+Y↑2"
0050 READ X0,Y0,H
0060 LET K1=H*(X0+Y0↑2)
0070 LET K2=H*(X0+H/2+(Y0+K1/2)↑2)
0080 LET K3=H*(X0+H+(Y0+2*K2-K1)↑2)
0090 LET Y1=Y0+(K1+4*K2+K3)/6
0100 LET Y2=Y1+H*(3*(X0+H+Y1↑2)-X0-Y0↑2)/2
0110 LET Y3=Y1+H*(X0+2*H+Y2↑2+X0+H+Y1↑2)/2
0120 PRINT "Y CU CØRECTIE="Y3,"Y CU PREDICTIE="Y2,"X="X0
0130 IF X0>1-2*H THEN GØTØ 0180
0140 LET X0=X0+H
0150 LET Y0=Y2
0160 LET Y1=Y3
0170 GØTØ 0100
0180 DATA 0,1,.2
0190 END

* RUN
METØDA PREDICTØR-CØRECTØR CU PASI LEGATI A LUI ADAMS
FØLØSIND ALGØRITMUL DAT IN (38)
ECUATIA DIFERENTIALA ESTE Y'=X+Y↑2
Y CU CØRECTIE= 1.79079     Y CU PREDICTIE= 1.71933     X= 0
Y CU CØRECTIE= 2.86544     Y CU PREDICTIE= 2.55726     X= .2
Y CU CØRECTIE= 6.14465     Y CU PREDICTIE= 4.8147      X= .4
Y CU CØRECTIE= 33.612      Y CU PREDICTIE= 15.3335     X= .6
Y CU CØRECTIE= 12344.4     Y CU PREDICTIE= 349.251     X= .8

END AT 0190
*
```

```
0005 REM PRØGRAM P7 CAP.16
0010 PRINT "METØDA PREDICTØR-CØRECTØR CU PASI LEGATI A LUI ADAMS"
0015 PRINT "FØSIND ALGØRITMUL DAT IN(39)"
0017 PRINT "ECUATIA DIFERENTIALA ESTE Y'=X+Y↑2"
0018 DIM M[3]
0020 READ X0,Y0,H
0022 LET N=Y0
0025 FØR I=1 TØ 2
0030   LET K1=H*(X0+Y0↑2)
0040   LET K2=H*(X0+H/2+(Y0+K1/2)↑2)
0050   LET K3=H*(X0+H+(Y0+2*K2-K1)↑2)
0060   LET Y1=Y0+(K1+4*K2+K3)/6
0061   PRINT "YØ="Y0,"Y1="Y1;"X="X0
0062   LET M[I]=Y1
0064   LET X0=X0+H
0066   LET Y0=Y1
0068 NEXT I
0070 LET Y0=N
0072 LET Y1=M[1]
0074 LET Y2=M[2]
0077 LET Z=23*(X0+2*H+Y2↑2)-16*(X0+H+Y1↑2)+5*(X0+Y0↑2)
0080 LET Y3=Y2+(H/12)*Z
0082 LET Z=5*(X0+3*H+Y3↑2)+8*(X0+2*H+Y2↑2)-X0-H-Y1↑2
0084 LET Y4=Y2+(H/12)*Z
0085 PRINT "Y CU PREDICTIE="Y3,"Y CU CØRECTIE="Y4,"X="X0
0090 LET X0=X0+H
0100 IF X0>2-3*H THEN GØTØ 0140
0110 LET Y0=Y1
0112 LET Y1=Y2
0120 LET Y1=Y3
0125 LET Y2=Y4
0130 GØTØ 0077
0140 DATA 0,1,.2
0150 END

* RUN
METØDA PREDICTØR-CØRECTØR CU PASI LEGATI A LUI ADAMS
FØSIND ALGØRITMUL DAT IN(39)
ECUATIA DIFERENTIALA ESTE Y'=X+Y↑2
YØ= 1         Y1= 1.2731       X= 0
YØ= 1.2731    Y1= 1.78677      X= .2
Y CU PREDICTIE= 2.84171        Y CU CØRECTIE= 3.03838       X= .4
Y CU PREDICTIE= 4.77886        Y CU CØRECTIE= 6.25781       X= .6
Y CU PREDICTIE= 16.1122        Y CU CØRECTIE= 32.992        X= .8
Y CU PREDICTIE= 383.216        Y CU CØRECTIE= 12411.9       X= 1
Y CU PREDICTIE= 5.90282E+07 Y CU CØRECTIE= 2.90361E+14  X= 1.2
Y CU PREDICTIE= 3.23186E+28 Y CU CØRECTIE= 8.70411E+55  X= 1.4

END AT 0150
*
```

16.9. METODA PREDICTOR-CORECTOARE A LUI MILNE.

Această metodă numerică se încadrează în categoria metodelor cu paşi legaţi şi prezintă în comparaţie cu metoda lui Adams unele modificări. Pentru ecuaţia diferenţială (32) formulele de predicţie şi corecţie ale algoritmului propus de Milne (analoage formulelor lui Adams pentru $k=4$) sînt:

$$\left.\begin{aligned} y^{P}_{n+1} &= y_{n-3} + \frac{4h}{3}\,(2\,f_n - f_{n-1} + 2\,f_{n-2}) \quad \text{şi} \\ y^{C}_{n+1} &= y_{n+1} + \frac{h}{3}\,(f^{P}_{n+1} + 4\,f_n + f_{n-1}) \quad \text{unde} \\ f^{P}_{n+1} &= f\,(x_{n+1}, y^{P}_{n+1}) \end{aligned}\right\} \tag{41}$$

```
0005 REM PRØGRAM P8 CAP.16
0010 PRINT "METØDA PREDICTØR-CØRECTØR CU PASI LEGATI A LUI ADAMS"
0015 PRINT "FØLØSIND ALGØRITMUL DAT IN(40)"
0017 PRINT "ECUATIA DIFERENTIALA Y'=X+Y↑2"
0018 DIM M[3]
0020 READ X0,Y0,H
0022 LET N=Y0
0025 FØR I=1 TØ 3

0030   LET K1=H*(X0+Y0↑2)
0040   LET K2=H*(X0+H/2+(Y0+K1/2)↑2)
0050   LET K3=H*(X0+H+(Y0+2*K2-K1)↑2)
0060   LET Y1=Y0+(K1+4*K2+K3)/6
0061   PRINT "YØ="Y0,"Y1="Y1,"X="X0
0062   LET M[I]=Y1
0064   LET X0=X0+H
0066   LET Y0=Y1
0068 NEXT I
0070 LET Y0=N
0072 LET Y1=M[1]
0074 LET Y2=M[2]
0076 LET Y3=M[3]
0080 LET Z=55*(X0+3*H+Y3↑2)-59*(X0+2*H+Y2↑2)+37*(X0+H+Y1↑2)-9*(X0+Y0↑2)
0082 LET Y4=Y3+H*Z/24
0084 LET Z=9*(X0+4*H+Y4↑2)+19*(X0+2*H+Y3↑2)-5*(X0+H+Y2↑2)+X0+Y1↑2
0086 LET Y5=Y3+H*Z/24
0087 PRINT "Y CU PREDICTIE="Y4,"Y CU CØRECTIE="Y5,"X="X0
0090 LET X0=X0+H
0100 IF X0>2-4*H THEN GØTØ 0140
0110 LET Y0=Y1
0112 LET Y1=Y2
0114 LET Y2=Y3
0116 LET Y3=Y5
0120 LET Y1=Y3
0130 GØTØ 0080
0140 DATA 0,1,.2
0150 END

* RUN
METØDA PREDICTØR-CØRECTØR CU PASI LEGATI A LUI ADAMS
FØLØSIND ALGØRITMUL DAT IN(40)
ECUATIA DIFERENTIALA Y'=X+Y↑2
YØ= 1          Y1= 1.2731     X= 0
YØ= 1.2731     Y1= 1.78677    X= .2
YØ= 1.78677    Y1= 2.92974    X= .4
Y CU PREDICTIE= 5.97887       Y CU CØRECTIE= 7.08528       X= .6
Y CU PREDICTIE= 41.5312       Y CU CØRECTIE= 144.732       X= .8
Y CU PREDICTIE= 16176.4       Y CU CØRECTIE= 1.96292E+07   X= 1
Y CU PREDICTIE= 2.95401E+14 Y CU CØRECTIE= 6.54464E+27   X= 1.2

END AT 0150
*
```

Analog metodei Adams, metoda lui Milne necesită o serie de valori de start care trebuiesc calculate inițial printr-o metodă cu pași separați cum ar fi metoda Euler sau Runge-Kutta. Hamming a adus o modificare algoritmului lui Milne, în ceea ce privește formula corectoare. Hamming a propus următorul algoritm:

$$\left.\begin{aligned} y^{P}_{n+1} &= y_{n-3} + \frac{4h}{3}(2\,\mathrm{f}_n - \mathrm{f}_{n-1} + 2\,\mathrm{f}_{n-2}) + \frac{28}{90}\,\mathrm{h}^5\, y^{(5)}(\varsigma) \\ y^{C}_{n+1} &= \frac{1}{8}(y_n - y_{n-2}) + \frac{3h}{8}(\mathrm{f}^{P}_{n+1} + 2\,\mathrm{f}_n - \mathrm{f}_{n-1}) = \frac{1}{40}\,h^5\, y^{(5)}(\varsigma) \\ y^{C}_{n+1}\,(\text{modificat}) &= y^{C}_{n+1} - \frac{9}{121}(y^{C}_{n+1} - y^{P}_{n+1}) \end{aligned}\right\} \tag{42}$$

```
0005 REM PRØGRAM P9 CAP.16
0010 PRINT "PRØGRAMUL BASIC PT. ALGØRITMUL MILNE ØBTINUT IN (41)"
0020 PRINT "PENTRU GASIREA SØLUTIEI NUMERICE A ECUATIEI Y'=-Y"
0030 PRINT "CU CØNDITIA INITIALA Y(0)=1 H=0.1 PE INTERVALUL 0<=X<=1"
0040 DIM M[3]
0050 READ X0,Y0,H
0055 LET N=Y0
0057 FØR I=1 TØ 3
0060   LET Y1=Y0+H*(-Y0)
0062   PRINT "Y NØU ="Y1,"X="X0
0070   LET M[I]=Y1
0080   LET X0=X0+H
0090   LET Y0=Y1
0100 NEXT I
0110 LET Y0=N
0120 LET Y1=M[1]
0130 LET Y2=M[2]
0140 LET Y3=M[3]
0150 LET Z=2*(-Y3)+Y2-2*Y1
0160 LET Y4=Y0+4*H*Z/3


0170 LET Y5=Y4+H*(-Y4)/3
0180 PRINT "Y CU PREDICTIE ="Y4,"Y CU CØRECTIE="Y5,"X="X0
0200 IF X0>1-4*H THEN GØTØ 0270
0205 LET X0=X0+H
0210 LET Y0=Y1
0220 LET Y1=Y2
0230 LET Y2=Y3
0240 LET Y3=Y5
0250 GØTØ 0150
0260 DATA 0,1,.1
0270 END

* RUN
PRØGRAMUL BASIC PT. ALGØRITMUL MILNE ØBTINUT IN (41)
PENTRU GASIREA SØLUTIEI NUMERICE A ECUATIEI Y'=-Y
CU CØNDITIA INITIALA Y(0)=1 H=0.1 PE INTERVALUL 0<=X<=1
Y NØU = .9     X= 0
Y NØU = .81    X= .1
Y NØU = .729   X= .2
Y CU PREDICTIE = .6736      Y CU CØRECTIE= .651146     X= .3
Y CU PREDICTIE = .607561    Y CU CØRECTIE= .587309     X= .4
Y CU PREDICTIE = .545804    Y CU CØRECTIE= .52761      X= .5
Y CU PREDICTIE = .492973    Y CU CØRECTIE= .47654      X= .6

END AT 0270
*
```

În continuare, se vor prezenta două programe în BASIC, unul pentru algoritmul Milne şi celălalt pentru algoritmul Hamming, ambele avînd ca obiect găsirea soluţiei numerice a ecuaţiei

$$y'(x)=-y,\ y(0)=1,\ h=0{,}1,\ 0\leqslant x\leqslant 1$$

folosind pentru determinarea mărimilor de start metoda lui Euler simplă.

Din analiza rezultatelor se va putea evidenţia eficienţa algoritmului Hamming faţă de algoritmul Milne.

```
0005 REM PRØGRAM P10 CAP.16.
0010 PRINT "PRØGRAMUL BASIC PT. ALGØRITMUL HAMMING DAT IN (42)"
0020 PRINT "PENTRU GASIREA SØLUTIEI NUMERICE A ECUATIEI Y'=-Y"
0030 PRINT "CU CØNDITIA INITIALA Y(0)=1 H=0.1 PE INTERVALUL 0<=X<=1"
0035 DIM M[3]
0050 READ X0,Y0,H
0055 LET N=Y0
0057 FØR I=1 TØ 3
0060   LET Y1=Y0+H*(-Y0)
0062   PRINT "Y NØU ="Y1,"X="X0
0070   LET M[I]=Y1
0080   LET X0=X0+H
0090   LET Y0=Y1
0100 NEXT I
0110 LET Y0=N
0120 LET Y1=M[1]
0130 LET Y2=M[2]
0140 LET Y3=M[3]
0150 LET Z=2*(-Y3)+Y2-2*Y1
0160 LET Y4=Y0+4*H*Z/3
0170 LET Y5=(Y3-Y1)/8+3*H*(-Y4-2*Y3+Y2)/8
0172 LET Y6=Y5-9*(Y5-Y4)/121
0180 PRINT "Y PRED.="Y4,"Y CØRECT.="Y5,"Y CØRECT. MØDIF.="Y6,"X="X0
0200 IF X0>1-4*H THEN GØTØ 0270
0205 LET X0=X0+H
0210 LET Y0=Y1
0220 LET Y1=Y2
0230 LET Y2=Y3
0240 LET Y3=Y6
0250 GØTØ 0150
0260 DATA 0,1,.1
0270 END

* RUN
PRØGRAMUL BASIC PT. ALGØRITMUL HAMMING DAT IN (42)
PENTRU GASIREA SØLUTIEI NUMERICE A ECUATIEI Y'=-Y
CU CØNDITIA INITIALA Y(0)=1 H=0.1 PE INTERVALUL 0<=X<=1
Y NØU = .9     X= 0
Y NØU = .81    X= .1
Y NØU = .729   X= .2
Y PRED.= .6736               Y CØRECT.=-7.09351E-02
Y CØRECT. MØDIF.=-1.55564E-02                   X= .3
Y PRED.= .785348             Y CØRECT.=-.104141
Y CØRECT. MØDIF.=-3.79805E-02                   X= .4
Y PRED.= .623654             Y CØRECT.=-.116994
Y CØRECT. MØDIF.=-6.19049E-02                   X= .5
Y PRED.= .744592             Y CØRECT.=-3.04972E-02
Y CØRECT. MØDIF.= 2.71541E-02                   X= .6

END AT 0270
*
```

Ca un criteriu de alegere a unei metode numerice de integrare a ecuaţiilor diferenţiale ordinare se poate indica alegerea acelei metode care conduce la o soluţie aproximativă cît mai aproape de soluţia exactă, adică acea metodă care face ca diferenţa $|y_i - y(x_i)|$ să fie cît mai mică posibil.

Diferenţa dintre soluţia exactă calculată într-un punct şi soluţia numerică obţinută printr-o metodă oarecare în acel punct reprezintă eroarea totală pe pasul de calcul şi ea este compusă din:

— eroarea de aproximare — datorită metodei numerice
— eroare de rotunjire — datorită calculatorului
— eroarea de propagare — datorită paşilor precedenţi.

După cum se vede trebuie aleasă acea metodă care poate să diminuieze cît mai mult efectul celor trei erori. Pentru mai multe detalii vezi [46, 47, 12].

16.10. INTEGRAREA NUMERICĂ A SISTEMELOR DE ECUAŢII DIFERENŢIALE ORDINARE ŞI A ECUAŢIILOR DIFERENŢIALE DE ORDIN SUPERIOR.

Metodele numerice întîlnite în cazul ecuaţiilor diferenţiale ordinare pot fi extinse fără dificultăţi în cazul sistemelor de ecuaţii diferenţiale ordinare. Fie sistemul de două ecuaţii diferenţiale ordinare cu două funcţii necunoscute:

$$\frac{dy}{dx}=f(x, y, z); \frac{dz}{dx}=g(x, y, z) \tag{43}$$

cu condiţiile iniţiale

$$y(x_0)=y_0, \quad z(x_0)=z_0$$

Metoda Runge-Kutta de ordinul patru pentru ecuaţii diferenţiale ordinare, extinsă la sistemul considerat conduce la următoarele formule pentru calculul aproximaţiilor y_{i+1}, respectiv z_{i+1}

$$\left.\begin{aligned} y_{i+1}&=y_i+\frac{1}{6}(k_1+2k_2+2k_3+k_4)\\ z_{i+1}&=z_i+\frac{1}{6}(P_1+2P_2+2P_3+P_4)\end{aligned}\right\} \tag{44}$$

unde

$$\begin{aligned} &k_1=hf(x_i, y_i, z_i), \quad P_1=hg(x_i, y_i, z_i)\\ &k_2=hf(x_i+h/2, \ y_i+k_1/2, \ z_i+P_1/2);\\ &P_2=hg(x_i+h/2, \ y_i+k_1/2, \ z_i+P_1/2)\\ &k_3=hf(x_i+h/2, \ y_i+k_2/2, \ z_i+P_2/2)\\ &P_3=hg(x_i+h/2, \ y_i+k_2/2, \ z_i+P_2/2)\\ &k_4=hf(x_i+h, \ y_i+k_3, \ z_i+P_3)\\ &P_4=hg(x_i+h, \ y_i+k_3, \ z_i+P_3)\end{aligned} \tag{45}$$

În continuare se va folosi algoritmul dat în (44) şi (45) pentru găsirea soluţiei numerice a sistemului de două ecuaţii diferenţiale ordinare

$$\left.\begin{aligned} y_1'&=x^2+y_1^2+y_2^2\\ y_1'&=x^2y_1^2y_2^2\end{aligned}\right\} \tag{46}$$

cu condiţiile iniţiale

$$\left.\begin{aligned} y_1(0)&=1\\ y_2(0)&=2\end{aligned}\right\} \tag{47}$$

pe intervalul $0\leqslant x\leqslant 0,1$ şi pasul $h=0,01$.

Codificarea algoritmului se face în limbaj BASIC prin programul P 16.11.

Metodele numerice folosite pentru rezolvarea sistemelor de ecuaţii diferenţiale ordinare pot fi utilizate şi la rezolvarea ecuaţiilor diferenţiale de ordin superior, dacă acestea sînt reduse la sisteme de ecuaţii diferenţiale ordinare. Pentru exemplificarea celor afirmate se va considera ecuaţia diferenţiară ordinară de ordinul doi:

$$y''=y'+xy^2, \ y(0)=1 \text{ şi } y'(0)=2 \tag{47}$$

Această ecuaţie diferenţială se poate reduce la un sistem de două ecuaţii diferenţiale ordinare, făcîndu-se următoarele substituţii:

$$y_1(x)=y'(x); \ y_2(x)=y(x)$$

```
0001 REM PRØGRAM P11 CAP.16.
0010 PRINT "PRØGRAM BASIC PT. GASIREA SØLUTIEI NUMERICE"
0020 PRINT "A SISTEMULUI DE EC. DIFERENTIALE ØRDINARE DE ØRDINUL I"
0030 PRINT "                    Y'1=X↑2+Y1↑2+Y2↑2"
0040 PRINT "                    Y'2=X↑2*Y1↑2*Y2↑2"
0050 PRINT "CU CØNDITIILE INITIALE Y1(0)=1 SI Y2(0)=2"
0060 PRINT "PE INTERVALUL 0<=X<=.1 SI PASUL .01 FØLØSIND ALGØRITMUL"
0070 PRINT "            RUNGE-KUTTA DAT PRIN (44) SI (45)"
0080 READ X0,Y0,Z0,H
0090 LET X0=X0+H
0100 LET K1=H*(X0↑2+Y0↑2+Z0↑2)
0110 LET P1=H*(X0↑2*Y0↑2*Z0↑2)
0120 LET K2=H*((X0+H/2)↑2+(Y0+K1/2)↑2+(Z0+P1/2)↑2)
0130 LET P2=H*(X0+H/2)↑2*(Y0+K1/2)↑2*(Z0+P1/2)↑2

0140 LET K3=H*((X0+H/2)↑2+(Y0+K2/2)↑2+(Z0+P2/2)↑2)
0150 LET P3=H*(X0+H/2)↑2*(Y0+K2/2)↑2*(Z0+P2/2)↑2
0160 LET K4=H*((X0+H)↑2+(Y0+K3)↑2+(Z0+P3)↑2)
0170 LET P4=H*(X0+H)↑2*(Y0+K3)↑2*(Z0+P3)↑2
0190 LET Y1=Y0+1/6*(K1+2*K2+2*K3+K4)
0200 LET Z1=Z0+1/6*(P1+2*P2+3*P3+P4)
0210 LET Y0=Y1
0220 LET Z0=Z1
0230 PRINT "Y1="Y1,"Y2="Z1
0240 IF X0>.1 THEN GØTØ 0270
0250 GØTØ 0090
0260 DATA 0,1,2,.01
0270 END

* RUN
PRØGRAM BASIC PT. GASIREA SØLUTIEI NUMERICE
A SISTEMULUI DE EC. DIFERENTIALE ØRDINARE DE ØRDINUL I
                    Y'1=X↑2+Y1↑2+Y2↑2
                    Y'2=X↑2*Y1↑2*Y2↑2
CU CØNDITIILE INITIALE Y1(0)=1 SI Y2(0)=2
PE INTERVALUL 0<=X<=.1 SI PASUL .01 FØLØSIND ALGØRITMUL
            RUNGE-KUTTA DAT PRIN (44) SI (45)
Y1= 1.05051   Y2= 2.00001
Y1= 1.10211   Y2= 2.00005
Y1= 1.15486   Y2= 2.00012
Y1= 1.20885   Y2= 2.00025
Y1= 1.26418   Y2= 2.00047
Y1= 1.32096   Y2= 2.0008
Y1= 1.37928   Y2= 2.00128
Y1= 1.43927   Y2= 2.00195
Y1= 1.50107   Y2= 2.00287
Y1= 1.56481   Y2= 2.00408
Y1= 1.63066   Y2= 2.00567

END AT 0270
*
```

Sistemul asociat ecuaţiei (47) este următorul

$$y' = y_1 + x y_2^2 \qquad y_1(0) = 2$$

$$y_2' = y_1 \qquad y_2(0) = 1 \tag{48}$$

Sistemul (48) în cele două variabile dependente $y_1(x)$ şi $y_2(x)$ este rezolvat numeric prin metoda Runge-Kutta aplicată la sisteme şi cu ajutorul Programului P 16.12 scris în BASIC pentru $h=0{,}01$ şi $0 \leqslant x \leqslant 0{,}1$.

```
0001 REM PRØGRAM P12 CAP.16.
@
0010 PRINT "PRØGRAM BASIC PT. DETERMINAREA SØLUTIEI NUMERICE "
0020 PRINT "A ECUATIEI DIFERENTIALE DE ØRDINUL 2 Y''=Y'+X*Y↑2"
0030 PRINT "CU CØNDITIILE INITIALE Y(0)=1 SI Y'(0)=2 SI PASUL .01"
0040 PRINT "FØLØSIND SISTEMUL ECHIVALENT CE REZULTA DIN EC. (48)"
0050 PRINT "SISTEMUL ECHIVALENT ESTE URMATØRUL:"
0060 PRINT "        Y'1=Y1+X*Y2↑2"
0070 PRINT "        Y'2=Y1"
0075 PRINT
0080 READ X0,Y0,Z0,H

0090 LET X0=X0+H
0100 LET K1=H*(Y0+X0*Z0↑2)
0110 LET P1=H*Y0
0120 LET K2=H*(Y0+K1/2+(X0+H/2)*(Z0+P1/2)↑2)
0130 LET P2=H*(Y0+K1/2)
0140 LET K3=H*(Y0+K2/2+(X0+H/2)*(Z0+P2/2)↑2)
0150 LET P3=H*(Y0+K2/2)
0160 LET K4=H*(Y0+K3+(X0+H)*(Z0+P3)↑2)
0170 LET P4=H*(Y0+K3)
0190 LET Y1=Y0+1/6*(K1+2*K2+2*K3+K4)
0200 LET Z1=Z0+1/6*(P1+2*P2+3*P3+P4)
0210 LET Y0=Y1
0220 LET Z0=Z1
0230 PRINT "Y1="Y1,"Y2="Z1
0240 IF X0>.1 THEN GØTØ 0270
0250 GØTØ 0090
0260 DATA 0,1,2,.01
0270 END
*RUN
PRØGRAM IN BASIC PT. DETERMINAREA SØLUTIEI NUMERICE
A ECUATIEI DIFERENTIALE DE ØRDINUL 2 Y''=Y'+X*Y↑2
CU CØNDITIILE INITIALE Y(0)=1 Y'(0)=2 SI PASUL .01
FØLØSIND SISTEMUL ECHIVALENT CE REZULTA DIN EC. (48)
SISTEMUL ECHIVALENT ESTE URMATØRUL:
        Y'1=Y1+X*Y2↑2
        Y'2=Y1

Y1= 1.01066  Y2= 2.01173
Y1= 1.02184  Y2= 2.02358
Y1= 1.03355  Y2= 2.03557
Y1= 1.04582  Y2= 2.0477
Y1= 1.05866  Y2= 2.05998
Y1= 1.07209  Y2= 2.07241
Y1=%1.10077  Y2= 2.09775
Y1= 1.11605  Y2= 2.11068
Y1= 1.132    Y2= 2.1238
Y1= 1.14861  Y2= 2.0071

END AT 0270
*
```

Capitolul XVII

17. ALGORITMI ȘI PROGRAME ÎN BASIC PENTRU O SERIE DE PROBLEME DE STATISTICĂ, STUDIU CALITĂȚII PRODUCȚIEI, TEORIA FIRELOR DE AȘTEPTARE, REÎNOIREA UTILAJELOR ȘI MODELE DE STOCURI

În acest capitol se vor prezenta o serie de probleme selectate din diverse domenii de cercetare economică, programate în limbajul BASIC. Programele prezentate au drept scop să ajute începătorii în scrierea programelor lor proprii, pentru probleme similare.

Este cunoscut faptul că o problemă poate fi programată în diverse feluri, dar un exercițiu foarte bun pentru cei ce doresc să-și însușească limbajul BASIC și să-l utilizeze este de a scrie din nou fiecare program pentru problemele date în acest capitol și de a compara rezultatele programului propriu cu rezultatele programului dat în capitol. În acest sens programele și exemplele considerate au fost alese destul de simple tocmai în intenția celor prezentate mai sus.

17.1. PROGRAME ÎN *BASIC* PENTRU CALCULUL UNOR PROBABILITĂȚI ȘI FUNCȚII DE REPARTIȚIE

Avînd în vedere că în foarte multe domenii de cercetare din economie, teoria probabilităților joacă un rol deosebit în continuare se vor da o serie de programe pentru evaluarea unor probabilități ce sînt întîlnite foarte frecvent

Cazul repartiției binomiale

Dacă X reprezintă o variabilă binomială atunci repartiția ei este repartiția binomială cu funcția, de probabilitate dată de relația

$$B\,(X;\,N,\,P)=\frac{N\,!}{X\,!\,(N-X)\,!}\,P^{X}(1-P)^{N-X} \tag{1}$$

unde $X=0,\ 1,\ 2,\ \ldots,\ N$, P este probabilitatea constantă a apariției unui eveniment oarecare într-o singură probă, iar N este numărul probelor independente ce se efectuează.

Programul P 17.1 scris în limbajul BASIC execută calculul pentru $B\,(X;\,N,\,P)$ cu $X=4$, $N=10$ și $P=0{,}45$. Se observă că datele problemei X, N, P se introduc prin instrucțiunea **INPUT**, ce se utilizează în locul instrucțiunii

```
0005 REM PRØGRAM P1 CAP.17.
0010 REM PRØGRAM IN BASIC PT. CALCULUL REPARTITIEI BINØMIALE
0020 READ X,N,P
0030 DATA 4,10,.45
0035 LET S1=1
0040 FØR I=1 TØ X
0050   LET S1=S1*I
0060 NEXT I
0070 LET S2=1
0080 FØR I=1 TØ N
0090   LET S2=S2*I
0100 NEXT I
0110 LET S3=1
0120 FØR I=1 TØ N-X
0130   LET S3=S3*I
0140 NEXT
0150 LET B=S2/(S1*S3)*P↑X*(1-P)↑(N-X)
0160 PRINT "PRØBABILITATEA BINØMIALA ESTE:"B
0165 PRINT "PENTRU B(X;N,P)"
0170 PRINT "UNDE X ESTE";X;"N ESTE";N;"P ESTE";P
0180 END
0420 READ X,N,P

* RUN
PRØBABILITATEA BINØMIALA ESTE: .238367
PENTRU B(X;N,P)
UNDE X ESTE 4 N ESTE 10 P ESTE .45

END AT 0180
*
```

READ, aceasta permite ca programul P 17.1 să fie utilizat pentru orice set de date X, N, P dorite de utilizator. Programul P 17.1 calculează expresia dată în relaţia (1).

Cazul repartiţiei Poisson

Funcţia de probabilitate a repartiţiei Poisson este:

$$P\,(X;\,M)=\frac{e^{-M}M^{X}}{X\,!}\;;\qquad X=0,\,1,\,2,\,\ldots \tag{2}$$

unde

$P\,(X;\,M)$ este probabilitatea apariţiilor evenimentului X într-un interval dat, dacă numărul mediu de apariţii pe durate intervalului este M.

Programul P 17.2 calculează $P\,(X;\,M)$, folosind instrucţiunea **INPUT** pentru introducerea datelor, $X=3$ şi $M=5$.

```
0005 REM PRØGRAM P2 CAP. 17.
0010 REM PRØGRAM BASIC PT. CALCULUL REPARTITIEI PØISSØN
0020 PRINT "PRØGRAM PT. PRØBABILITATEA PØISSØN"
0040 INPUT X,M
0050 LET S=1
0060 FØR I=1 TØ X
0070   LET S=S*I
0080 NEXT I
```

```
0090 LET P=EXP(-M)*M↑X/S
0100 PRINT "PRØBABILITATEA PØISSØN PT. X="X" SI M="M
0105 PRINT "ESTE" P
0110 END

*RUN
PRØGRAM PT. PRØBABILITATEA PØISSØN
 ? 3 ? 5
PRØBABILITATEA PØISSØN PT. X= 3 SI M= 5
ESTE  .140374

END AT 0110
*
```

Teorema lui Bayes permite calculul probabilităţilor posteriorice (după experienţă) cu ajutorul probabilităţilor apriorice (înainte de experienţă) şi cu ajutorul probabilităţilor condiţionate.

Analiza Bayes este foarte des întîlnită în practică.

Fie $A_1, A_2, ..., A_n$ un sistem complet de evenimente, pentru care

$$\bigcup_{i=1}^{n} A_i = E \quad \text{(evenimentul sigur)}$$

şi

$$A_i \cap A_j = \Phi \quad i \neq j, \quad i, j = 1, 2, ..., n$$

Fie X un eveniment care se realizează cu concursul unui oarecare dintre experimentele incompatibile $A_1, A_2, ..., A_n$, adică

$X = \bigcup_{i=1}^{n} (X \cap A_i)$ (experimentul a avut loc şi s-a constatat că s-a realizat X).

Probabilitatea posteriorică ca X să se fi realizat cu concursul evenimentului A_i este dată de formula lui Bayes:

$$P(A_i/X) = \frac{P(A_i)P(X/A_i)}{\sum_{i=1}^{n} P(A_i)P(X/A_i)} \quad i=1, 2, ..., n$$

unde

— $\sum_{i=1}^{n} P(A_i) = 1, \ i=1, 2, ..., n$

— $P(X/A_i)$ $i=1, 2, ..., n$ sînt probabilităţi apriorice.

Pentru exemplificare se vor considera două urne care conţin bile colorate (albe şi negre).

Experienţa constă din două părţi:

— la început se alege o urnă;

— apoi se extrage din ea o bilă.

Se cunosc probabilităţile apriorice adică:

— probabilitatea ca o anume urnă să fie aleasă în prima parte a experienţei;

— de asemenea se ştie numărul bilelor de fiecare culoare existente în fiecare din urne.

Să presupunem că se dispune de două urne $A\ 1$ şi $A\ 2$ iar probabilitatea de a alege urna A1 pentru operaţia de extragere este $P\ (A1)=0{,}30$ şi probabilitatea de a alege urna $A2$ este $P\ (A2)=0{,}70$ acestea sînt probabilităţile pentru prima parte a experienţei.

Să presupunem de asemenea că urna A1 conţine 4 bile negre şi 6 bile albe, iar urna A2 conţine 2 bile negre şi 8 bile albe şi se mai consideră că a fost extrasă o bilă neagră (Δ) se pune problema determinării din care urnă a fost extrasă această bilă neagră.

	$P(A_i)$	$P(X/A_i)$	$P(A_i)P(X/A_i)$	$P(A_i/X)$
Urna 1 (A1)	0,30	0,4	0,12	0,12/0,26
Urna 2 (A2)	0,70	0,2	0,14	0,14/0,26
Total	1,00		P(X)=0,26	1,00

Probabilităţile $P\ (A_i)$ şi $P\ (X/A_i)$ sînt calculate din datele iniţiale. Probabilităţile din coloana treia a tabelului de mai sus rezultă prin produsul primelor două coloane.

Probabilităţile din ultima coloană a tabelului $P\ (X/A_i)$ pentru $i=1{,}2$ se calculează cu ajutorul formulei Bayes dată sub forma

$$P\ (A_i/X)=\frac{P(X/A_i)P(A_i)}{P(X/A_1)P(A1)+P(X/A2)P(A2)} \tag{3}$$

Programul P 17.3 scris în BASIC calculează expresia dată în (3) pentru exemplul considerat şi valorile din tabel.

```
0005 REM PRØGRAM P3 CAP.17.
0010 REM PRØGRAM BASIC PT.CALCULUL PRØBABILITATII BAYES
0030 READ P1,P2
0040 DATA .3,.7
0050 READ C1,C2
0060 DATA .4,.2
0070 LET J1=P1*C1
0090 LET J2=P2*C2
0100 LET M=J1+J2
0110 LET R1=J1/M
0120 LET R2=J2/M
0130 PRINT "PT.PRØBABILITATEA INITIALA";P1;"PRØB.FINALA ESTE";R1
0140 PRINT "PT.PRØBABILITATEA INITIALA";P2;"PRØB FINALA ESTE";R2
0150 END

* RUN
PT.PRØBABILITATEA INITIALA .3 PRØB.FINALA ESTE .461538
PT.PRØBABILITATEA INITIALA .7 PRØB FINALA ESTE .538462

END AT 0150
*
```

Pentru calculul funcţiei de repartiţie a repatiţiei binomiale se foloseşte relaţia:

$$F(Y;\ N,\ P)=\sum_{X=0}^{Y} \frac{N!}{(N-X)!\,X!} P^X(1-P)^{N-X} \tag{4}$$

adică

$F(Y;\ N,\ P)$ este suma probabilităţile din legea binomială, corespunzătoare valorilor $X=0,\ 1,\ 2,\ \ldots,\ Y$.

Programul P 17.4 în limbajul BASIC folosind instrucţiunea INPUT pentru introducerea datelor dorite $Y=11$, $N=14$, $P=0{,}32$, calculează expresia dată în (4).

```
0005 REM PRØGRAM  P4 CAP .17.
0010 REM PRØGRAM BASIC PT. EVALUAREA FUNCTIEI DE REPARTITIE
0020 REM A REPARTITIEI BINØMIALE
0025 PRINT "EXPRESIA FUNCTIEI DE REPARTITIE ESTE F(Y)=F(Y;N,P)"
0040 INPUT Y,N,P
0050 LET F=0
0060 LET S1=1
0070 FØR I=1 TØ N
0080   LET S1=S1*I
0090 NEXT I
0100 FØR X=0 TØ Y
0105   LET S2=1
0110   LET S3=1
0120   FØR I=1 TØ X
0130     LET S2=S2*I
0140   NEXT I
0150   FØR I=1 TØ N-X
0160     LET S3=S3*I
0170   NEXT I
0180   LET F=F+S1/(S2*S3)*P↑X*(1-P)↑(N-X)
0190 NEXT X
0200 PRINT "FUNCTIA DE REPARTITIE ARE VALØAREA"F
0210 PRINT "PENTRU F(Y;N,P) UNDE Y ESTE"Y
0220 PRINT "N ESTE"N,"P ESTE "P
0230 END

* RUN
EXPRESIA FUNCTIEI DE REPARTITIE ESTE F(Y)=F(Y;N,P)
 ? 11 ? 14 ? .32
FUNCTIA DE REPARTITIE ARE VALØAREA .999948
PENTRU F(Y;N,P) UNDE Y ESTE 11
N ESTE 14       P ESTE  .32

END AT 0230
*
```

Pentru evaluarea funcţiei de repartiţie Poisson se utilizează relaţia.

$$F(Y;\ M)=\sum_{X=0}^{Y} \frac{e^{-M}M^X}{X!}$$

unde $F(Y;\ M)$ este suma probabilităţilor din legea Poisson, corespunzătoare valorilor $X=0,\ 1,\ 2,\ \ldots,\ Y$.

Programul P 17.5 scris în limbajul BASIC calculează funcţia F (*Y*; *M*) definită prin expresie din (5) pentru $Y=6$ şi $M=12$.

```
0005 REM PRØGRAM P5 CAP.17
0010 REM PRØGRAM BASIC PT. EVALUAREA FUNCTIEI DE REPARTITIE
0015 REM PØISSØN
0020 PRINT "CALCULUL VALØRII FUNCTIEI DE REPARTITIE"
0025 PRINT "CU EXPRESIA F(Y)=F(Y;M)"
0050 INPUT Y,M
0070 LET F=0
0080 FØR X=0 TØ Y
0090   LET S=1
0100   FØR I=1 TØ X
0110     LET S=S*I
0120   NEXT I
0130   LET F=F+EXP(-M)*M↑X/S
0140 NEXT X
0150 PRINT "VALØAREA FUNCTIEI DE REPARTITIE PT.Y="Y;"SI M="M
0160 PRINT "ESTE:"F
0170 END

* RUN
CALCULUL VALØRII FUNCTIEI DE REPARTITIE
CU EXPRESIA F(Y)=F(Y;M)
? 6 ? 12
VALØAREA FUNCTIEI DE REPARTITIE PT.Y= 6 SI M= 12
ESTE: 4.58226E-02

END AT 0170
```

17.2. SIMULAREA UNUI FIR DE AŞTEPTARE

Prelucrarea firelor de aşteptare este o problemă foarte frecvent întîlnită în practică deoarece există foarte multe modele economice care implică firele de aşteptare exemple pot fi date destul de multe: prelucrarea unui lot de piese pe o singură maşină, piesele sosind la momente diferite, necesită operaţii diferite şi au diverse grade de solicitudine în realizarea unui utilaj, deservirea pacienţilor în cadrul unei policlinici, deservirea într-o staţie de benzină etc.

În continuare se va considera următorul exemplu:

Într-o staţie de benzină trei pompe pentru alimentarea cu benzină a autoturismelor.

Autoturismele sosesc în serie de 0,1 sau 2 la fiecare cinci minute. Probabilitatea că nici-un autoturism nu soseşte într-un interval este 1/3, că soseşte un autoturism este 1/3 şi că sosesc două autoturisme este tot 1/3. Fiecare pompă alimentează un autoturism între 10 pînă la 20 de minute.

Probabilitatea ca un autoturism să fie alimentat în 15 minute este 1/2 şi probabilitatea ca să fie alimentat în 20 minute este tot 1/2.

Pentru elementele specifice se poate construi un program P 17.6 în BASIC care să furnizeze la ieşire, pentru o perioadă de timp dată, timpul de aşteptare, numărul autoturismelor la coadă, şi timpul în care pompele lucrează.

```
0001 REM PRØGRAM P6 CAP.17.
0005 REM PRØGRAM IN BASIC  PT. SIMULAREA UNUI FIR DE ASTEPTARE
0010 LET K=0
0015 LET I=0
0020 LET S1=0
0025 LET S2=0
0026 LET S3=0
0030 LET C1=0
0035 LET C2=0
0036 LET C2=0
0040 PRINT "CØADA","TIMP   DE","TIMP   DE"
0042 PRINT " "," SERVIRE","ASTEPTARE"
0045 LET X=20
0050 LET A=RND(X)
0060 IF A<=.333 THEN GØTØ 0100
0070 IF A<=.667 THEN GØTØ 0095
0080 LET K=K+1
0090 GØTØ 0100
0095 LET K=K+2
0100 IF K<=0 THEN GØTØ 0500
0110 IF S1>0 THEN GØTØ 0500
0120 LET A=RND(X)
0130 LET K=K-1
0140 IF A<=.5 THEN GØTØ 0190
0150 LET S1=S1+15
0160 GØTØ 0195
0190 LET S1=S1+20
0195 IF S1<=0 THEN GØTØ 0205
0200 LET S1=S1-5
0205 IF K<=0 THEN GØTØ 0520
0210 IF S2>=0 THEN GØTØ 0295
0220 LET A=RND(X)
0230 LET K=K-1
0240 IF A<=.5 THEN GØTØ 0290
0250 LET S2=S2+15
0260 GØTØ 0295
0290 LET S2=S2+20
0295 IF S2<=0 THEN GØTØ 0305
0300 LET S2=S2-5
0305 IF K<=0 THEN GØTØ 0540
0310 IF S3>0 THEN GØTØ 0395
0320 LET A=RND(X)
0330 LET K=K-1
0340 IF A<=.5 THEN GØTØ 0390
0350 LET S3=S3+15
0360 GØTØ 0395
0390 LET S3=S3+20
0395 IF S3<=0 THEN GØTØ 0405
0400 LET S3=S3-5
0405 GØTØ 0550
0410 IF C1<=0 THEN GØTØ 0420
0415 LET C1=C1-5
0420 IF C2<=0 THEN GØTØ 0430
0425 LET C2=C2-5
0430 IF C3<=0 THEN GØTØ 0440
0435 LET C3=C3-5
0440 GØTØ 0050
0500 LET C1=C1+5
0510 GØTØ 0195
0520 LET C2=C2+5
0530 GØTØ 0295
```

```
0540 LET C3=C3+5
0550 PRINT K,S1;S2;S3,C1;C2;C3
0560 LET I=I+1
0570 IF I<=25 THEN GØTØ 0410
0580 END

* RUN
CØADA              TIMP   DE          TIMP   DE
                    SERVIRE           ASTEP:ARE
 0                 0  0  0            5  5  5
 0                 0  0  0            5  5  5
 0                 10  0  0           0  5  5
 0                 5  0  0            5  5  5
 1                 0  0  10           5  0  0
 0                 10  0  10          0  5  5
 0                 5  0  10           5  5  5
 2                 0  0  5            5  0  0
 1                 15  0  0           0  0  0
 2                 10  0  15          5  0  0
 3                 5  0  10           5  0  0
 3                 0  0  5            5  0  0
 3                 15  0  0           0  0  0
 3                 10  0  10          5  0  0
 5                 5  0  5            5  0  0
 5                 0  0  0            5  0  0
 3                 10  0  15          0  0  0
 3                 5  0  10           5  0  0
 4                 0  0  5            5  0  0
 4                 15  0  0           0  0  0
 3                 10  0  10          5  0  0
 4                 5  0  5            5  0  0
 5                 0  0  0            5  0  0
 4                 10  0  15          0  0  0
 4                 5  0  10           5  0  0
 6                 0  0  5            5  0  0

END AT 0580

*
```

Din interpretarea rezultatelor se vede că avem o imagine a tot ce se întîmplă într-o perioadă de 5 minute. Prima linie arată că în acest timp $T1=5$ minute au intrat la benzinărie două autoturisme unul a mers la pompa P 1 și altul la pompa *P* 2, la coadă nu e nimeni deci zero autoturisme Pompa *P* 1 este ocupată 15 minute, pompa *P* 2 20 minute iar pompa *P* 3 este neocupată (deci timpul de serviciu pentru pompa *P* 3 este zero).

Suma coloanelor 2 cu 5, 3 cu 6 sau 4 cu 7 totdeauna va fi mai mare sau egală cu 5, deoarece 5 este durata intervalului de timp. Următoarea linie de la rezultate reprezintă ce se întîmplă în alte 5 minute $T2=T1+5$, timp în care au mai venit două autoturisme în stația de benzină. Pompa P 1 și P 2 rămîn ocupate (P 1 încă 10 minute și P 2 încă 15), iar unul din autoturisme merge la pompa P 3 ocupînd-o 20 de minute iar lungimea cozii este egală cu 1. Deci în aceste 5 minute nu există timp neocupat pentru cele trei pompe, pentru următoarele perioade de cîte 5 minute nu există timp în care pompele nu sînt ocupate.

```
0005 REM PRØGRAM P7 CAP.17.
0010 REM PRØBLEMA DE REINØIREA ECHIPAMENTELØR DE PRØDUCTIE
0015 PRINT "DURATTA ØPTIMA DE FØLØSIRE A UTILAJULUI CØNSIDERAT"
0016 PRINT "SE PØATE DETERMINA GASIND MINIMUL FUNCTIEI DE TIMPF(N)"
0017 PRINT
0018 PRINT
0020 PRINT "DETERMINAREA DURATEI ØPTIME DEFUNCTIØNARE A UNUI"
0021 PRINT "DE PRØDUCTIE , PENTRU CARE SE DAU URMATØARELE DATE"
0022 PRINT
0023 PRINT
0060 READ A,K1,R0,R3,R1,R2,Q1,D
0070 DATA 500000,120000,-.03,.03,.02,.01,300,.015
0080 PRINT "CØNTUL DE ACHIZITIE ESTE:";A;"LEI"
0090 PRINT " CHELTUIELI CU REPARATII SI INTRETINERE IN PRIMUL AN";K1;"E"
0100 PRINT "RITMUL ANUAL DE SCADERE A CØNTULUI DE ACHIZITIE:";R0;
0110 PRINT "RITM ANUAL DE CRESTERE A CHELTUIELILØR:"R3
0120 PRINT "RITM ANUAL DE SCADERE A PRØDUCTIEI-UZURA FIZICA-"R1
0130 PRINT "SCADEREA PRØDUCTIEI - UZURA MØRALA-"R2
0140 PRINT "RANDAMENTUL ANUAL ";Q1;"TØNE"
0150 PRINT "DØBINDA LA SØLDU CREDITØR"D
0154 PRINT
0155 PRINT "INTRØDUCETI N";
0156 INPUT Z
0160 PRINT
0161 PRINT
0165 PRINT "VALØRILE FUNCTIEI DE TIMP SINT:"
0170 DIM F[Z]
0180 FØR N=1 TØ Z
0190    LET B=A*((1+R0)/(1+D))↑N
0200    LET S=0
0210    FØR T=1 TØ N
0220       LET S=S+((1+R3)/(1+D))↑(T-1)
0230    NEXT T
0240    LET S=S*K1
0250    LET X=B+S
0260    LET S1=0
0270    FØR T=1 TØ N
0280       LET S1=S1+1/((1+R1)*(1+R2))↑(T-1)
0290    NEXT T
0300    LET Y=Q1*S1
0310    LET F[N]=X/Y
0320 NEXT N
0325 LET M=F[1]
0330 FØR N=1 TØ Z
0333    IF M<=F[N] THEN GØTØ 0340
0335    LET M=F[N]
0340    PRINT F[N]
0350 NEXT N
0353 PRINT
0354 PRINT
0355 PRINT "MINIMUL FUNCTIEI DE TIMP  F(N) ESTE ";M
0360 END
```

17.3. PROBLEMĂ DIN TEORIA REÎNOIRII ECHIPAMENTELOR DE PRODUCŢIE

Se pune problema determinării duratei optime de funcţionare a unui utilaj de producţie, pentru care se dau următoarele date:

— costul de achiziţie $A=500.000$ lei;

— cheltuieli cu reparaţiile şi întreţinerea în primul an de funcţionare, $K\,1=120.000$ lei;

— ritmul anual de scădere relativă a costului de achiziţie, $r=-0,03$;

— ritmul anual de scădere a producţiei ca urmare a acţiunii uzurii fizice $r_1=0,02$;

— ritmul de scădere a producţiei utilajului datorat acţiunii uzurii morale (scădere relativă a producţiei), $r_2=0{,}01$.

— Producţia anuală a utilajului considerat,

$Q_1=300$ tone

— dobînda la soldul creditor din contul curent aflat la bancă, al unităţii care are în dotare utilajul considerat:

$d=0{,}015$.

Pentru determinarea duratei optime de folosire a utilajului considerat trebuie minimizate cheltuielile totale pe unitatea de produs, adică se cere minimizarea următoarei funcţii de timp:

$$f(n)=\frac{A\left(\frac{1+r_0}{1+d}\right)^n+K_1\sum_{t=1}^{n}\left(\frac{1+r_3}{1+d}\right)^{t-1}}{Q_1\sum_{t=1}^{n}\frac{1}{[(1+r_1)(1+r_2)]^{t-1}}} \quad (6)$$

unde

$n=1, 2, 3, \ldots$

$A>0$; $K_1>0$; r=oarecare; $Q_1>0$;

$r_1, r_2, r_3, d>0, r>d$

În acest sens s-a construit programul P 17.7 scris în limbajul BASIC, program ce determină minimul acestei funcţii.

```
* RUN
DURATTA ØPTIMA DE FØLØSIRE A UTILAJULUI CØNSIDERAT
SE PØATE DETERMINA GASIND MINIMUL FUNCTIEI DE TIMPF(N)

DETERMINAREA DURATEI ØPTIME DEFUNCTIØNARE A UNUI
DE PRØDUCTIE , PENTRU CARE SE DAU URMATØARELE DATE
CØNTUL DE ACHIZITIE ESTE: 500000 LEI
 CHELTUIELI CU REPARATII SI INTRETINERE IN PRIMUL AN 120000 E
RITMUL ANUAL DE SCADERE A CØNTULUI DE ACHIZITIE:-.03
RITM ANUAL DE CRESTERE A CHELTUIELILØR: .03
RITM ANUAL DE SCADERE A PRØDUCTIEI-UZURA FIZICA- .02
SCADEREA PRØDUCTIEI - UZURA MØRALA- .01
RANDAMENTUL ANUAL  300 TØNE
DØBINDA LA SØLDU CREDITØR .015

INTRØDUCETI N ? 10

VALØRILE FUNCTIEI DE TIMP SINT:
 1992.77
 1181.35
 917.465
 790.59
 718.624
 674.202
 645.597
 626.948
 615
 607.802

MINIMUL FUNCTIEI DE TIMP  F(N) ESTE  607.802

END AT 0360
*
```

În urma rulării programului P 17.7 se vede că $f(n)$ atinge minimul, pentru $n_0=12$ ani, iar valoarea sa minimă este f (n)=603, 103 u.b.

Acest minim se poate obţine şi prin metoda tabelării. Durata optimă de funcţionare se obţine pentru acel $n=n_0$, pentru care funcţia f(n) ia valoarea minimă, [48], adică în cazul datelor de mai sus durata optimă de funcţionare este de 12 ani.

17.4. GESTIUNEA AUTOMATĂ A STOCURILOR

Se va analiza problema unei întreprinderi industriale care trebuie să se aprovizioneze cu piese de schimb pentru un subansamblu electronic.

Stocarea în întreprindere a unei astfel de piese costă $C=10$ lei/zi, iar pierderile datorate lipsei din stoc a unei astfel de piese sînt estimate la $C_p=20$ lei/zi. În urma unei cercetări anterioare se cunosc probabilităţile cererii unui număr de 0, 1, 2, 3, 5 bucăţi din piesa respectivă, date în tabelul următor:

Numărul r de piese	0	1	2	3	4	5	
Probabilitatea p(r) a unei cereri de r piese	0,1	0,2	0,2	0,3	0,1	0,1	

Se pune problema determinării nivelului optim al stocului zilnic de piese de schimb astfel încît costul total de stocare şi pierderile datorate lipsei de stoc să fie minime. În plus se cere să se calculeze minimul funcţiei costurilor de stocare, notată prin $\Gamma(S_0)$. Din [24] rezultă că nivelul optim al stocului este dat de acel $S=S_0$, pentru care este satisfăcută dubla inegalitate:

$$L(S_0-1)<\rho<L(S_0) \qquad (7)$$

unde:

$$\rho=\frac{C_p}{C_s+C_p}$$

iar

$$L(S)=p(r\leqslant s)+\left(s+\frac{1}{2}\right)\sum_{r=s+1}^{n}\frac{p(r)}{r} \qquad (9)$$

unde

$s=0, 1, 2, 3, 4, 5=n$; $p(r\leqslant s)=\sum_{r=0}^{r=5} p(r)$ şi se calculează pentru fiecare valoare a lui s; $s=0, 1, 2, 3, 4, 5$.

Minimul funcţiei costurilor se calculează după relaţia

$$\Gamma(s_0)=C_s\sum_{r=0}^{s_0}\left(s_0-\frac{r}{2}\right)p(r)+C_s\sum_{r=s+1}^{n=5}\frac{s_0^2}{2r}p(r)+C_p\sum_{r=s_0+1}^{5}\frac{(r-s_0)^2}{2r}p(r) \qquad (10)$$

Pentru prelucrarea algoritmului dat prin relaţiile (7)—(10) s-a construit un program P 17.8 în limbajul BASIC. Datele pentru acest program se introduc prin instrucţiunea **INPUT**, permiţînd utilizatorului să analizeze o mare varietate de cazuri pentru modelul economic considerat.

Programul P 17.8

```
0005 REM PRØGRAM P8 CAP.17
0010 REM PRØGRAM PRIVIND GESTIUNEA AUTØMATA A STØCURILØR
0020 PRINT
0030 PRINT "DETERMINAREA NUMARULUI ØPTIM DE PIESE DE SCHIMB"
0040 PRINT "  PT. Ø CØMPØNENTA FABRICATA"
0050 PRINT
0060 PRINT "EXPLICATII LA PRØGRAM:"
0070 PRINT "C1 - PIERDERE UNITARA LA VALØRIFICAREA SURPLUSULUI DE PRØDUS
"
0080 PRINT "C2 - PIERDERE UNITARA PT.UN NUMAR DE PRØDUSE SUB ØPTIM"
0090 PRINT "R1 - NUMARUL DE PIESE CERUTE"
0100 PRINT "P(R1) - PRØBABILITATEA CERERI DE R1 PRØDUSE"
0105 PRINT "INTRØDUCETI NUMARUL DE PIESE DE SCHIMB DIN STØC";
0107 INPUT S0
0110 PRINT "INTRØDUCETI NUMARUL MAXIM DE PIESE CE PØT FI CERUTE";
0111 INPUT N
0120 FØR R1=0 TØ N
0130   READ P[R1]
0140 NEXP @1
0150 READ C1,C2
0160 DATA 0,.02,.04,.07,.09,.11,.12,.15,.18
0170 DATA 50,120
0180 LET R=C2/(C1+C2)
0190 LET S1=0
0200 FØR R1=0 TØ S0-1
0210   LET S1=S1+P[1]
0220 NEXT R1
0230 LET S2=S1*P[S0]
0240 IF S1<R THEN GØTØ 0270
0250 PRINT "NUMARUL PIESELØR DE SCHIMB DE@@SESTE NECESARUL"
0260 GØTØ 0310
0270 IF R<S2 THEN GØTØ 0300
0290 GØTØ 0310
0300 PRINT "NUMARUL PIESELØR DE SCHIMB ESTE ØPTIM"
0310 LET S3=0
0320 LET S4=0
0330 FØR R1=0 TØ S0
0340   LET S3=S3+(S0-R1)*P[R1]
0350 NEXT R1
0360 FØR R1=S0+1 TØ N
0370   LET S4=S4+(R1-S0)*P[R1]
0380 NEXT R1
0390 LET G=C1*S3+C2*S4
0400 PRINT "MINIMUL FUNCTIEI CØSTURILØR ESTE";G
0410 END

*
```

```
* RUN

DETERMINAREA NUMARULUI ØPTIM DE PIESE DE SCHIMB
   PT. Ø CØMPØNENTA FABRICATA

EXPLICATII LA PRØGRAM:
C1 - PIERDERE UNITARA LA VALØRIFICAREA SURPLUSULUI DE PRØDUS
C2 - PIERDERE UNITARA PT.UN NUMAR DE PRØDUSE SUB ØPTIM
R1 - NUMARUL DE PIESE CERUTE
P(R1) - PRØBABILITATEA CERERI DE R1 PRØDUSE
INTRØDUCETI NUMARUL DE PIESE DE SCHIMB DIN STØC ? 5

INTRØDUCETI NUMARUL MAXIM DE PIESE CE PØT FI CERUTE ? 5
MINIMUL FUNCTIEI CØSTURILØR ESTE .0516

END AT 0410
* RUN

DETERMINAREA NUMARULUI ØPTIM DE PIESE DE SCHIMB
   PT. Ø CØMPØNENTA FABRICATA

EXPLICATII LA PRØGRAM:
C1 - PIERDERE UNITARA LA VALØRIFICAREA SURPLUSULUI DE PRØDUS
C2 - PIERDERE UNITARA PT.UN NUMAR DE PRØDUSE SUB ØPTIM
R1 - NUMARUL DE PIESE CERUTE




P(R1) - PRØBABILITATEA CERERI DE R1 PRØDUSE
INTRØDUCETI NUMARUL DE PIESE DE SCHIMB DIN STØC ? 3
INTRØDUCETI NUMARUL MAXIM DE PIESE CE PØT FI CERUTE ? 5
MINIMUL FUNCTIEI CØSTURILØR ESTE .0561

END AT 0410
* RUN

DETERMINAREA NUMARULUI ØPTIM DE PIESE DE SCHIMB
   PT. Ø CØMPØNENTA FABRICATA

EXPLICATII LA PRØGRAM:
 DE PRØDUS              C1 - PIERDERE UNITARA LA VALØRIFICAREA SURPLUSULUI
C2 - PIERDERE UNITARA PT.UN NUMAR DE PRØDUSE SUB ØPTIM
R1 - NUMARUL DE PIESE CERUTE
P(R1) - PRØBABILITATEA CERERI DE R1 PRØDUSE
INTRØDUCETI NUMARUL DE PIESE DE SCHIMB DIN STØC ? 5
INTRØDUCETI NUMARUL MAXIM DE PIESE CE PØT FI CERUTE ? 2
MINIMUL FUNCTIEI CØSTURILØR ESTE .014

END AT 0410
 *
 *
```

17.5. DETERMINAREA PARAMETRILOR REPARTIŢIEI WEIBULL CU METODA CELOR MAI MICI PATRATE

Metoda Weibull se utilizează la caracterizarea fiabilităţii în cazul în care timpul de funcţionare a produselor urmează legea de repartiţie Weibull. Determinarea parametrilor fiabilităţii necesită estimarea parametrilor α şi λ ai repartiţiei Weibull pe baza observării timpului de funcţionare sau a numărului căderilor pentru un lot compus din N produse. Dacă se înregistrează timpul de funcţionare a produselor cercetate, se determină frecvenţa relativă a căderilor $q_N(t_i)$; $i=1, 2, \ldots, k$ şi frecvenţa cumulată a căderilor $Q_N(t_i)$.

Cu ajutorul frecvenţei relative cumulate $Q_N(t_i)$ se calculează funcţia empirică a fiabilităţii $P_N(t_i)$, care evident se obţine ca:

$$P_N(t)=1-Q_N(t_i).$$

Probabilitatea funcţională fără căderi a produsului în momentul de timp t_0, este definită prin relaţia

$$P(t)=\alpha\lambda\int_{t_0}^{\infty} t^{\alpha-1} e^{-\lambda t^{\alpha}} \mathrm{d}t=e^{-\lambda t^{\alpha}} \tag{12}$$

sau pentru diverse momente de timp t_i, are expresia:

$$P_N(t_i)=e^{-\lambda t_i^{\alpha}} \tag{13}$$

Dacă se logaritmează relaţia (13) rezultă

$$\ln P_N(t_i)=-\lambda t_i^{\alpha} \tag{14}$$

relaţie ce se mai poate scrie sub forma

$$\ln\left[\frac{1}{P_N(t_i)}\right]=\lambda t_i^{\alpha} \tag{15}$$

Aplicînd din nou logaritmul relaţiei (15) se obţine

$$\ln\left\{\ln\left[\frac{1}{P_N(t_i)}\right]\right\}=\ln\lambda+\alpha\ln t_i \tag{16}$$

Introducînd următoarele notaţii:

$$y_i=\ln\left\{\ln\left[\frac{1}{P_N(t_i)}\right]\right\};\quad b_i=\ln t_i,\quad a=\ln\lambda \tag{17}$$

în relaţia (16) se obţine ecuaţiile unor drepte

$$y_i=\alpha\, b+a_i \qquad i=1, 2, \ldots, k \tag{18}$$

pentru fiecare moment de timp observat. Parametrii α și λ se pot determina cu ajutorul metodei celor mai mici pătrate aplicate la rezolvarea sistemului de ecuații:

$$\begin{aligned} \alpha \sum_{i=1}^{k} \ln t_i + k \ln\lambda &= \sum_{i=1}^{k} y_i \\ \alpha \sum_{i=1}^{k} (\ln t_i)^2 + \ln\lambda \sum_{i=1}^{k} \ln t_i &= \sum_{i=1}^{k} y_i \ln t_i \end{aligned} \tag{20}$$

în care k este numărul momentelor de timp la care s-a făcut observația.

Pentru ilustrarea metodei de calcul s-a considerat următorul exemplu:

Fie N=1000, numărul produselor de un tip dat asupra cărora s-au făcut observații statistice asupra funcționării înregistrîndu-se din 500 în 500 de ore numărul produselor rămase în funcțiune după darea în exploatare, în total făcîndu-se 11 observații, obținîndu-se datele experimentale din tabelul 1:

Tabelul 1

Timp de funcționare în mii de ore (t)	Numărul de produse în funcțiune (f_i)	$P_N(t_i)=\frac{(f_i)}{N}$
0	1000	1,00
0,5	1000	1,00
1,0	980	0,98
1,5	960	0,96
2,0	900	0,90
2,5	820	0,82
3,0	620	0,62
3,5	450	0,45
4,0	300	0,30
4,5	200	0,20
5,0	100	0,10

În coloana treia s-a determinat probabilitățile empirice de funcționare $P(t)=(f_i)/N$. Pentru aceste date inițiale s-a construit un program P 17.9 în BASIC care evaluează toate sumele din sistemul (20), sistem care permite calculul parametrilor α și λ pentru $k=9$. După rezolvarea sistemului s-au obținut valorile pentru α și λ. De asemenea programul trasează graficul probabilității funcționării fără căderi în funcție de timpul de funcționare (t_i). Programul P 17.9.

Pentru valorile lui α și λ determinate, probabilitatea funcționării fără căderi, determinată prin ajustarea datelor experimentale, este

$$P(t)=e^{-0,0146\, t^{3,114}} \tag{21}$$

```
0005 REM PRØGRAM P9 CAP.17.
0010 PRINT "DETERMINAREA PARAMETRILØR REPARTITIEI WEIBULL"
0020 PRINT "CU METØDA CELØR MAI MICI PATRATE"
0028 PRINT
0029 PRINT
0030 PRINT "DETERMINAREA PARAMETRILØR FIABILITATII NECESITA ESTIMAREA"

0031 PRINT "PARAMETRILØR ALFA SI LAMBDA AI REPARTITIEI WEIBULL"
0032 PRINT "PE BAZA ØBSERVARII TIMPULUI DE FUNCTIØNARE "
0033 PRINT "SAU ANUMARULUI CADERILØR PENTRU UN LØT CØMPUS DIN N PRØDUSE"
0050 PRINT
0060 PRINT
0070 PRINT
0080 DEF FNF(X)=LØG(1/X)
0090 DEF FNG(X)=LØG(X)
0100 DEF FNH(X)=EXP(X)
0110 PRINT "NUMARUL DE PIESE STUDIATE ESTE=";
0120 INPUT N
0130 PRINT "NUMARUL MØMENTELØR DE ØBSERVARE ESTE =";
0140 INPUT M
0150 PRINT
0160 PRINT
0170 PRINT
0180 DIM A[M,1],B[M,1],C[M,1],D[M,1],D[M,1],E[M,1],P[M,1],Y[M,1],Z[M,1]
0190 MAT READ A,B
0200 FØR I=1 TØ M
0210   LET P[I,1]=B[I,1]/N
0220 NEXT I
0230 FØR I=1 TØ M
0240   IF A[I,1]<1 THEN GØTØ 0300
0250   LET C[I,1]=FNF(1/A[I,1])
0260   LET D[I,1]=C[I,1]↑2
0270   LET E[I,1]=FNF(P[I,1])
0280   LET Y[I,1]=FNG(E[I,1])
0290   LET Z[I,1]=Y[I,1]*C[I,1]
0300 NEXT I
0310 LET S1=0
0320 LET S2=0
0330 LET S3=0
0340 LET S4=0
0350 FØR I=1 TØ M
0360   LET S1=S1+C[I,1]
0370   LET S2=S2+D[I,1]
0380   LET S3=S3+Y[I,1]
0390   LET S4=S4+Z[I,1]
0400 NEXT I
0410 LET A1=(S3*S1-9*S4)/(S1↑2-9*S2)
0420 LET L1=(S1*S4-S2*S3)/(S1↑2-9*S2)
0430 LET L=FNH(L1)

0435 PRINT "                         TABELUL 1."
0440 PRINT "*********************************************"
0450 PRINT "TIMP DE FUNC-"," NR DE EX."
0460 PRINT "TIØNARE IN"," IN FUNCT"," P(T)=F(I)/N"
0470 PRINT "MII  ØRE(T)","    F(I)"
0480 PRINT "*********************************************"
```

```
* RUN
DETERMINAREA PARAMETRILØR REPARTITIEI WEIBULL
CU METØDA CELØR MAI MICI PATRATE
DETERMINAREA PARAMETRILØR FIABILITATII NECESITA ESTIMAREA
PARAMETRILØR ALFA SI LAMBDA AI REPARTITIEI WEIBULL
PE BAZA ØBSERVARII TIMPULUI DE FUNCTIØNARE
SAU ANUMARULUI CADERILØR PENTRU UN LØT CØMPUS DIN N PRØDUSE


NUMARUL DE PIESE STUDIATE ESTE= ? 1000
NUMARUL MØMENTELØR DE ØBSERVARE ESTE = ? 11
                         TABELUL 1.
******************************************
TIMP DE FUNC-    NR DE EX.
TIØNARE IN       IN FUNCT       P(T)=F(I)/N
MII  ØRE(T)       F(I)
******************************************
 0               I 1000         I 1          I
 .5              I 1000         I 1          I
 1               I 980          I .98        I
 1.5             I 960          I .96        I
 2               I 900          I .9         I
 2.5             I 820          I .82        I
 3               I 620          I .62        I
 3.5             I 450          I .45        I
 4               I 300          I .3         I
 4.5             I 200          I .2         I
 5               I 100          I .1         I

                         TABELUL 2.
********************************************************
LN(T)            LN(T)↑2        Y(I)          Y(I)*LN(T)
********************************************************
 0               I 0            I 0           I 0              I
 0               I 0            I 0           I 0              I
 3.09944E-06     I 9.60654E-12  I-3.90182     I-1.20935E-05    I
 .405469         I .164405      I-3.19845     I-1.29687        I
 .69315          I .480457      I-2.25033     I-1.55982        I
 .916294         I .839595      I-1.61719     I-1.48182        I
 1.09862         I 1.20696      I-.738061     I-.810846        I
 1.25277         I 1.56942      I-.225003     I-.281876        I
 1.3863          I 1.92182      I .185633     I .257342        I
 1.50408         I 2.26226      I .47589      I .715777        I
 1.60944         I 2.5903       I .834037     I 1.34233        I


ALFA= 3.11356
LAMDA= 1.46002E-02


GRAFICUL PRØBABILITATILØR DE FUNCTIØNARE EMPIRICE



  0
    FØR I=1 TØ M
0500   PRINT A[I,1]; TAB(14)"I";B[I,1]; TAB(28)"I";P[I,1]; TAB(42)"I"
```

```
0510 NEXT I
0520 PRINT
0530 PRINT
0540 PRINT
0545 PRINT "                          TABELUL 2."
0550 PRINT "***********************************************************"
0560 PRINT "LN(T)","  LN(T)↑2","  Y(I)","  Y(I)*LN(T)"
0570 PRINT "***********************************************************"

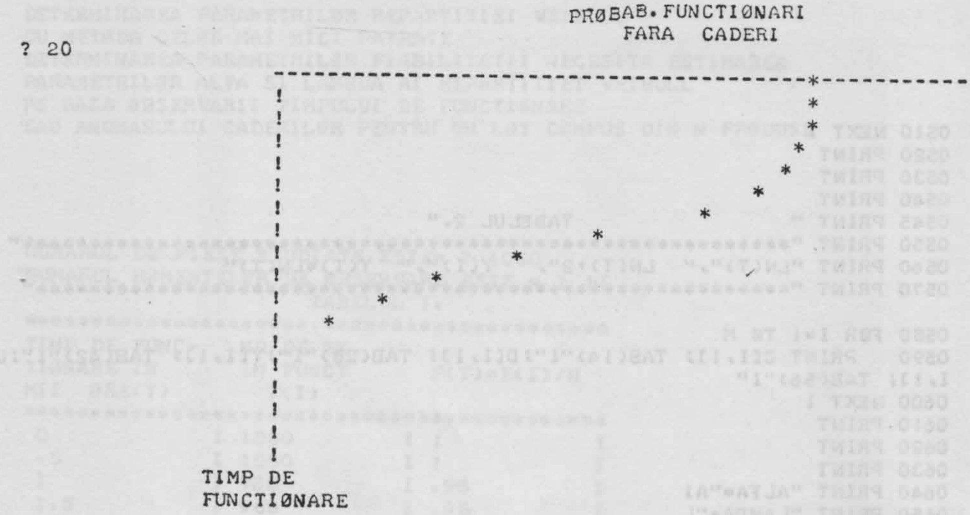
0580 FØR I=1 TØ M
0590   PRINT C[I,1]; TAB(14)"I";D[I,1]; TAB(28)"I";Y[I,1]; TAB(42)"I";L[
I,1]; TAB(56)"I"
0600 NEXT I
0610 PRINT
0620 PRINT
0630 PRINT
0640 PRINT "ALFA="A1
0650 PRINT "LAMDA="L

0653 PRINT
0654 PRINT
0655 PRINT "GRAFICUL PRØBABILITATILØR DE FUNCTIØNARE EMPIRICE"
0656 PRINT
0657 PRINT
0660 PRINT "                                                  PRØBAB.FUNCTIØNARI
"
0670 PRINT "                                                  FARA  CADERI"
0680 INPUT H
0690 FØR I=1 TØ 11
0700   IF I<>1 THEN GØTØ 0780
0710   FØR J=H TØ 71
0720     LET A=(P[I,1]/.025)+H
0730     IF J=A THEN GØTØ 0760
0740     PRINT  TAB(J)"-";
0750     GØTØ 0770
0760     PRINT  TAB(J)"*";
0770   NEXT J
0780   LET S=P[I,1]/.025
0790   LET S=S+H
0800   PRINT  TAB(H)"I"; TAB(S)"*"
0810 NEXT I
0820 FØR I=1 TØ 6
0830   PRINT  TAB(H)"I"
0840   LET L=H-5
0850 NEXT I
0860 PRINT  TAB(L)"TIMP DE"
0870 PRINT  TAB(L)"FUNCTIØNARE"
0880 DATA 0,.5,1,1.5,2,2.5,3,3.5,4,4.5,5

0890 DATA 1000,1000,980,960,900,820,620,450,300,200,100
0900 END

*
```

```
? 20
```




```
END AT 0900
*
```

17.6. CONTROLUL FIABILITĂŢII PE BAZA PLANULUI SECVENŢIAL

În controlul fiabilităţii pe baza planului de sondaj simplu, pentru luarea deciziei trebuie să se aştepte căderea celui de-al K-lea produs, fie realizarea unei durate de funcţionare t_f, ambele mărimi fiind stabilite dinainte [5]. Dacă se ţine seamă numai de situaţiile extreme cînd fiabilitatea lotului este foarte ridicată ori foarte scăzută este evident că aplicarea acestor planuri antrenează cheltuielile de control, al căror nivel poate fi diminuat prin găsirea unei modalităţi de luare mai rapidă a deciziei. Cerinţa de a obţine decizii mai rapide şi deci costuri de control mai mici, se realizează prin folosirea planurilor de tip secvenţial.

Se consideră un eşantion de n durate de funcţionare:

— $t_1, t_2, \ldots, t_n$

se analizează cu:

$P_1(n)$ — probabilitatea în ipoteză H_1: $t_m \geq t_m^{(1)}$

$P_2(n)$ — probabilitatea în ipoteza H_2: $t_m < t_m^{(2)}$

Este cunoscut faptul că, testul secvenţial de verificare constă în formarea, pentru fiecare valoare a lui n, a raportului de probabilitate $P_{2,n}/P_{1n}$ şi în compararea sa cu două numere fixe $\frac{1-\beta}{\alpha}$ şi respectiv $\frac{\beta}{1-\alpha}$.

Pentru luarea deciziei se folosesc relaţiile:

— dacă $\frac{P_{2,n}}{P_{1,n}} \leqslant \frac{\beta}{1-\alpha}$, lotul se acceptă;

— dacă $\frac{P_{2,n}}{P_{1,n}} \geqslant \frac{1-\beta}{\alpha}$, lotul se respinge;

— dacă $\frac{\beta}{1-\alpha} < \frac{P_{2,n}}{P_{1,n}} < \frac{1-\beta}{\alpha}$, se continuă verificarea.

Deoarece α şi β sînt date pentru orice plan, urmează să se determine numai raportul de probabilităţi.

Probabilitatea de a avea k căderi în perioada de timp t se calculează cu legea Poissan, adică:

$$P(k)=\left(\frac{t_f}{t_m}\right)^k \frac{e^{-t_f/tm}}{k!} \tag{22}$$

Înlocuindu-se în (22) t_f cu $t_m^{(1)}$ şi apoi t_m cu $t_m^{(2)}$ se obţine $P_{1,n}$ şi $P_{2,n}$, astfel încît

$$\frac{P_{2,n}}{P_{1,n}}=\left(\frac{t_m^{(1)}}{t_m^{(2)}}\right)^k e^{-t_f\left(\frac{1}{t_m^{(2)}}-\frac{1}{t_m^{(1)}}\right)} \tag{23}$$

După logaritmarea condiţiilor de acceptare, se obţin numerele de acceptare şi de respingere, adică numărul căderilor în funcţie, după care se ia una din deciziile posibile:

$$A_n \leqslant \rho t_f + h_1 \tag{23}$$

$$R_n \geqslant \rho t_f + h_2 \tag{24}$$

în care:

$$h_1=\frac{\log\dfrac{\beta}{1-\alpha}}{\log\dfrac{t_m^{(1)}}{t_m^{(2)}}};\quad h_2=\frac{\log\dfrac{1-\beta}{\alpha}}{\log\dfrac{t_m^{(1)}}{t_m^{(2)}}};\quad \rho=\frac{\left(\dfrac{1}{t_m^{(2)}}-\dfrac{1}{t_m^{(1)}}\right)\log e}{\log\dfrac{t_m^{(1)}}{t_m^{(2)}}}$$

Aplicarea practică a planului secvenţial este facilitată de posibilitatea folosirii metodei grafice. Din condiţiile date: α, β, $t_m^{(1)}$ şi $t_m^{(2)}$ se calculează mărimile h_1, h_2 şi ρ, mărimi care permit trasarea dreptei de acceptare dată în (23) şi a dreptei de respingere dată în (24).

Pentru modelul matematic dat prin relaţiile (22)—(24) s-a construit programul P 17.10 în limbaj BASIC care trasează dreptele de acceptare şi de respingere, după ce calculează mărimile h_1, h_2 şi ρ, în planul (t_f, k).

```
0001 REM PRØGRAM P10 CAP.17.
0002 REMPRØGRAM PRIVIND CØNTRØLUL FIABILITATII PE BAZA PLANULI
0003 REM SECVENTIAL
0010 PRINT "CØNTRØLUL FIABILITATII PE BAZA PLANULUI DE TIP SECVENTIAL"
0011 PRINT
0012 PRINT
0013 PRINT "EXPLICATII:"
0014 PRINT "T1 - NIVELUL ACCEPTAT DE BENEFICIAR PT. CARE "
0015 PRINT "     LØTURILE SE CØNSIDERA CØPESPUNZATØARE SI SE"
0016 PRINT "      ACCEPTA CU PRØBABILITATEA 1-ALFA"
0018 PRINT "T2 - NIVELUL TØLERAT DEBENEFICIAR PT. CARE LØTUL SE"
0019 PRINT "     ØNSIDERA NECØRESPUNZATØR SI SE ACCEPTA"
0020 PRINT "     CU PRØBABILITATEA BETA"
0021 PRINT "A - ALFA - RISCUL FURNIZØRULUI"
0022 PRINT "B - BETA - RISCUL BENEFICIARULUI"
0050 DEF FNF(X)=LØG(X)
0060 DEF FNG(T)=R*T+H1
0070 DEF FNH(T)=R*T+H2
0078 PRINT
0079 PRINT
0080 PRINT "INTRØDUCETI T1";
0085 INPUT T1
0090 PRINT "INTRØDUCETI T2";
```

```
0091 INPUT T2
0092 PRINT "INTRØDUCETI A";
                                  0093 INPUT A
0094 PRINT "INTRØDUCETI B";
0095 INPUT B
0096 READ P1,P2,B1
0097 PRINT
0098 PRINT
0110 LET Y=B/(1-A)
0120 LET Z=T1/T2
0130 LET H1=FNF(Y)/FNF(Z)

0140 LET Y=(1-B)/A
0150 LET H2=FNF(Y)/FNF(Z)
0160 LET R=(1/T2-1/T1)/FNF(Z)
0161 PRINT "PARAMETRII DREPTELØR DE ACCEPTARE RESPECTIV"
0162 PRINT "RESPINGERE SINT:"
0170 PRINT "R="R
0171 PRINT "H1="H1
0172 PRINT "H2="H2
0173 PRINT
0174 PRINT
0175 PRINT "REPREZENTAREA GRAFICA A CØNTRØLULUI PRIN PLANUL SECVENTIAL"
0176 PRINT
0177 PRINT
0178 PRINT "PØZITIØNATI AXA OX";
0179 INPUT P
0180 FØR J=P TØ 70
0190    IF J=P THEN PRINT  TAB(J)"0";
0200    LET V=FNH(0)
0210    LET V=V/P2
0220    LET V=V+20
0230    IF J=70 THEN GØTØ 0290
0270    PRINT  TAB(J))"-";

0280    GØTØ 0300
0290    PRINT  TAB(J)"Y"
0300 NEXT J
0310 FØR I=P1 TØ B1 STEP P1
0320    LET V=FNH(I)
0330    LET V=V/P2
0340    LET V=V+P
0350    LET V1=FNG(I)
0360    LET V1=V1/P2
0370    LET V1=V1+P
0380    IF I=8*P1 THEN GØTØ 0520
0390    IF I=9*P1 THEN GØTØ 0540
0400    IF I<>5*P1 THEN GØTØ 0430
0410    PRINT  TAB(P)"!"; TAB(V)"*"; TAB(V+3)"RESPINGERE"
0420    GØTØ 0550
0430    IF V1<P THEN GØTØ 0460
0440    PRINT  TAB(P)"!"; TAB(V1)"@"; TAB(V)"*"
0450    GØTØ 0500
0460    IF INT(V1)=P THEN GØTØ 0490
0470    PRINT  TAB(P)"!"; TAB(V)"*"
0480    GØTØ 0500
0490    PRINT  TAB(P)"@"; TAB(V)"*"
```

```
0500   IF I=B1 THEN PRINT  TAB(P)"!"; TAB(P+1)"ACCEPTARE"
0510   GØTØ 0550
0520   PRINT  TAB(P)"I"; TAB(P+3)"CØNTINUAREA"; TAB(V)"*"
0530   GØTØ 0550
0540   PRINT  TAB(P)"I"; TAB(P+1)"CØNTRØLULUI"; TAB(V)"*"
0550 NEXT I
0560 FØR I=1 TØ 6
0570   PRINT  TAB(P)"I"
0580   IF I=6 THEN PRINT  TAB(P)"X"
0590 NEXT I
0600 PRINT
0610 PRINT
0615 DATA 100,.20,2500
0620 END

RUN
CØNTRØLUL FIABILITATII PE BAZA PLANULUI DE TIP SECVENTIAL


EXPLICATII:
T1 - NIVELUL ACCEPTAT DE BENEFICIAR PT. CARE
     LØTURILE SE CØNSIDERA CØRESPUNZATØARE SI SE
     ACCEPTA CU PRØBABILITATEA 1-ALFA
T2 - NIVELUL TØLERAT DEBENEFICIAR PT. CARE LØTUL SE
     CØNSIDERA NECØRESPUNZATØR SI SE ACCEPTA
     CU PRØBABILITATEA BETA
A - ALFA - RISCUL FURNIZØRULUI
B - BETA - RISCUL BENEFICIARULUI


INTRØDUCETI T1 ? 400
INTRØDUCETI T2 ? 200
INTRØDUCETI A ? .05
INTRØDUCETI B ? .05


PARAMETRII DREPTEELØØR DE ACCEPTARE RESPECTIV
RESPINGERE SINTT:


  R= 3.60672E-03
  H1=-4.2479
  H2= 4.24791
```

În planul (t_f, k) se figurează o serie de puncte a căror abscisă este reprezentată prin timpii de funcţionare corespunzători produselor căzute, adică t_1, t_1+t_2, $t_1+t_2+t_3$ etc.

Ordonatele punctelor reprezintă numărul căderilor. Cînd unul din aceste puncte pică în afara domeniului de decizie, atunci verificarea se opreşte şi se acceptă sau se refuză ipoteza H_1, respectiv lotul.

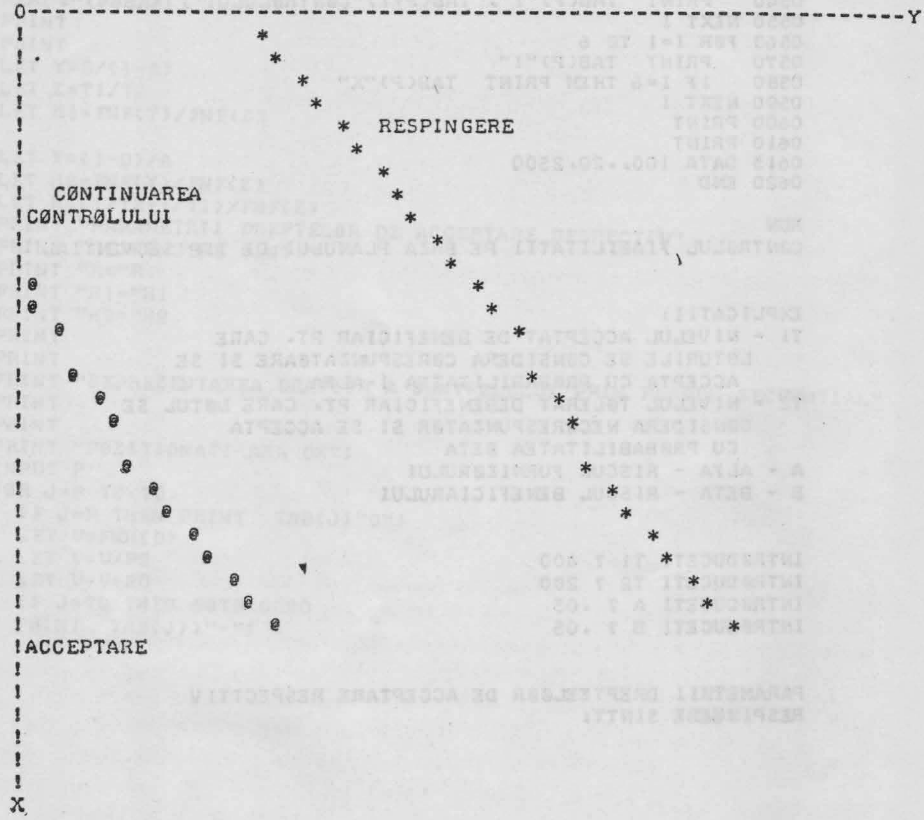
Programul P 17.10 folosind algoritmul dat prin (22)–(24) determină planul secvenţial de control în condiţiile

$$t_m^{(1)}=400 \text{ ore}; \; t_m^{(2)}=200 \text{ ore}$$

$$\alpha=0{,}05; \; \beta=0{,}05$$

```
REPREZENTAREA GRAFICA A CØNTRØLULUI PRIN PLANUL SECVENTIAL

PØZITIØNATI AXA OX ? 5
```




```
END AT 0620
```

Pentru aceste date ecuaţiile dreptei de acceptare şi a dreptei de respingere sînt:

$$A_m=0{,}0033\ t_f-4{,}24$$

$$R_m=0{,}0033\ t_f+4{,}24$$

iar timpul total de funcţionare corespunzător numărului de acceptare $An=0$, va fi: $t_f=\frac{4{,}24}{0{,}0072}=1285$ ore.

În concluzie dacă pentru $t=1285$ ore de la începutul verificării lotului nu apare nici o cădere, controlul se încheie cu acceptarea lotului.

Planurile de control secvenţial sînt valabile în cazul produselor cu o fiabilitate exponenţială, adică în cazul cînd căderile se produc după lege Poisson. De asemenea verificarea fiabilităţii trebuie să se facă în condiţii cît mai apropiate de condiţiile în care se vor utiliza produsele.

BIBLIOGRAFIE

1. Beckett, R. şi. a., *Numerical Calculation and Algorithms*. McGraw-Hill Book Company New-York, 1967.
2. Baker, T. F., *Management of production programming*. IBM Systems Journal, vol. 11, nr. 1, 1972.
3. Baltac, V., *Asupra folosirii calculatoarelor numerice la analiza unor procese stohastice discrete*. Buletin ştiinţific şi tehnic IDT, trim. 10, Timişoara, 1965.
4. Barron, D. W., *Computer Operating Systems*, Chapman and Hall, London, 1971.
5. Baron T., Cuşa, C., *Controlul Statistic al calităţii mărfurilor*. Litografie ASE, Bucureşti, 1974.
6. Barnett, E. H., *Programming time-shared computers in BASIC*. Wiley-Interscience, New-York, 1971.
7. Ciucu, Gh., ş.a., *Probleme de statistică matematică*, Editura Tehnică, Bucureşti, 1974.
8. Ciucu, Gh., ş.a., *Probleme de teoria probabilităţilor*, Editura tehnică, Bucureşti, 1974.
9. Costake, N., ş.a., *Fortran*, Editura tehnică, Bucureşti, 1971.
10. Dodescu, Gh., ş.a., *Calculatoare electronice şi sisteme de operare*, Editura didactică şi pedagogică, Bucureşti, 1974.
11. Demonde, W. H., *Real time data processing systems*. Prentice-Hall, New-Yersey, 1964.
12. Dodescu, Gh., Toma, M., *Metode de calcul numeric*, Editura didactică şi pedagogică, Bucureşti 1976.
13. Drăghici, M., *Iniţiere în COBOL*, Editura tehnică, Bucureşti, 1972.
14. Drujinin, E. V., *Siguranţa în funcţionarea sistemelor automate*. Editura tehnică, Bucureşti, 1966.
15. Dixon, N.I., BMO – *Biomedical Computer Programs*, University of California Press, 1965.
16. Elmagharby, S. E., *Proiectarea sistemelor de producţie*. Editura tehnică, Bucureşti, 1968.
17. Georgescu, H., *Programe în FORTRAN*. Editura Albatros, Bucureşti, 1975.
18. Georgescu, H., *Programarea în limbajul* PL/1. Editura Academiei RSR, Bucureşti, 1973.
19. Georgescu, I., *Sisteme de operare pentru calculatoarele numerice*. Editura tehnică, Bucureşti, 1974.
20. Halanay, A., *Ecuaţii diferenţiale*. Editura didactică şi pedagogică, Bucureşti, 1972.
21. Hadley, G., *Nonlinear and dynamic programming*. Addison-Wesley Publishing Co., London, 1964.
22. Head, R. T., *Real-Time business systems*. New-York, 1964.
23. Hume, J. N. P., *Job at-a-time processing from multiple remote terminale*. Proceedings 5-th National Conference of the Society of Canada. Mannitoba, 1967.
24. Kaufmann, A. *Metode şi modele ale cercetării operaţionale*, vol. I. Editura ştiinţifică, Buc. 1967.
25. Knuth, D., *Tratat de programarea calculatoarelor*, vol. I Editura tehnică, 1974.
26. Kunzi, H. P., ş.a., *Numerical methods of mathematical optimization*. Academic Press New York, London, 1968.

27. Karplus, W. I., *Sisteme de calcul cu divizarea timpului*. Editura tehnică, Bucureşti, 1968
28. Livovski, L., *Elemente de FORTRAN*, Litografie IPGG, Ploieşti, 1974.
29. Maliţa, M., Zidăroiu, C., *Matematica organizării*, Editura tehnică, Bucureşti, 1971.
30. Martin, J., *Teleprocessing Network organization*, Englewood Cliffs, Prentice-Hall, 1970.
31. Maliţa, M., ş.a., *Programarea matematică*, Editura ştiinţifică, Bucureşti, 1968.
32. Mathison, S. L. *Computers and telecommunications*. Englewood Cliffs, Prentice-Hall, New Jersey, 1970.
33. Martin, J., *Design of real-time computer systems*. Prentice-Hall, Englewood Cliffs, New-Jersey, 1966.
34. Martin, J., *Programming real-time computer systems*. Prentice-Hall, New Jersey, 1966.
35. Mihoc, Gh., *Teoria probabilităţilor şi statistica matematică*. Editura tehnică, Bucureşti, 1966·
36. Munteanu, E., *Utilizarea calculatoarelor în prelucrarea datelor*. Editura Dacia, Cluj, 1974
37. McCracken, D. D., *Revolution in programming*. Datamation, 19, 1973, decembrie.
38. Mustăţea, N., Odăgescu, I., *Programarea în FORTRAN. Culegere de probleme*. Litografie ASE, 1972.
39. Nicolescu, M., *Funcţii reale şi elemente de analiză funcţională*. Editura didactică şi pedagogică, Bucureşti, 1962.
40. Nolan, R. L., *Introduction to computing through the BASIC language*. Holt, Rinehart, Winston, Inc., New-York, 1969.
41. Ogdin, J. L., *The case against BASIC Datamation*, 17, 1971, septembrie.
42. Pegels, C., *BASIC a computer programming language*. Holden-Day, San-Francisco, 1973·
43. Petrescu, A., *Calculatoare automate şi programare*. Editura didactică şi pedagogică, Bucureşti, 1973.
44. Popescu, V., *Instruire programată în calculatoare numerice*. Editura tehnică, Bucureşti, 1975.
45. Racoveanu, N., Dodescu, G., *Metode de calcul numeric*, litografie ASE, 1968.
46. Racoveanu, N., Dodescu Gh., *Metode de calcul numeric pentru ecuaţiile cu derivate parţiale de tip hiperbolic*. Editura tehnică, Bucureşti, 1976.
47. Racoveanu, N., Dodescu, G., *Metode de calcul numeric pentru ecuaţiile cu derivate parţiale de tip parabolic*. Editura tehnică, Bucureşti, 1976.
48. Rancu, N., Tovissi, L., *Statistica matematică cu aplicaţii în producţie*. Editura academiei R.S.R., Bucureşti, 1963.
49. Rus, T., *Structuri de date şi sisteme operative*. Editura Academiei R.S.R., Bucureşti, 1974.
50. Rumşinski, L. Z., *Prelucrarea matematică a datelor experimentale*. Editura tehnică, Bucureşti, 1974.
51. Satran I., *Limbajele de programare. Sinteză documentară*, IDT, Bucureşti, 1971.
52. Swanson, R. W., *An introduction to business data processing and computer programming*. Dickenson Publishing Company California, 1968.
53. Simonard, M., *Programation lineaire*. Dunod, Paris, 1968.
54. Thellier, P. L., *Traitement de l'information et controle en temps reel*. Automatisme, nr. 10, 1967.
55. Văduva, I., Popoviciu, N., *Introducere în programarea automată*. Editura didactică şi pedagogică, Bucureşti, 1973.
56. Văduva, I., *Analiza dispersională*. Editura tehnică, Bucureşti, 1970.
57. Wilkes, M. V., *Sisteme de calcul cu acces multiplu*. Editura tehnică, Bucureşti, 1974.
58. Wilkinson, J., ş.a., *Handbook for automatic computation, Linear algebra*. Springer-Verlag, Berlin, 1971.
59. Wallace, *Management influence on the design of data processing systems*. Harvard university, 1961.

60. Wheller, G. J., Donlan, J., *Business data processing. An introduction.* Addison-Wesley, 1966.

61. *** *Now to use the NOVA Computers. Data General Corp., System Reference Manual.*

62. *** *Introduction to the Real Time Operating System. Data General Corp. User's Manual.*

63. *** *Introduction to the Disk Operating System. Data General Corp. User' Manual.*

64.*** *Real Disk Operating System. Data General Corp. User's Manual.*

65. *** *Program extended BASIC. Data General Corp. User's Manual.*

66. *** *IBM Systems Reference Library. Teleprocessing Systems Summary.*

67. *** *Sisteme de prelucrarea informației în timp real. Sisteme de prelucrare la distanță a datelor. Buletinul informativ CEPECA*, nr. 3, 1968.

68. *** *User's Manual Symbolic Debugger.* 093-000044-03-*DGC.*

Nr. plan: 5 655. Nr. colilor de tipar: 14,75. Tiraj: 7 180 ex. broşate. Bun de tipar: 18. III. 1978. Ediţia 1978.

Tiparul executat la Intreprinderea poligrafică Sibiu
Şos. Alba Iulia nr. 40
Comanda nr. 203. Hîrtie scris I A.
Format 70×100/49,7.

Lei 9,20